

SpECTRE: A Task-based Discontinuous Galerkin Code for Relativistic Astrophysics

Lawrence E. Kidder^a, Scott E. Field^a, Francois Foucart^b, Erik Schnetter^{c,d,e}, Saul A. Teukolsky^a, Andy Bohn^a, Nils Deppe^a, Peter Diener^{e,f}, François Hébert^a, Jonas Lippuner^g, Jonah Miller^{d,c}, Christian D. Ott^g, Mark A. Scheel^g, Trevor Vincent^{h,i}

^aCornell Center for Astrophysics and Planetary Science, Cornell University, Ithaca, New York 14853, USA

^bLawrence Berkeley National Laboratory, 1 Cyclotron Rd, Berkeley, CA 94720, USA; NASA Einstein Fellow

^cPerimeter Institute for Theoretical Physics, Waterloo, ON, Canada

^dDepartment of Physics, University of Guelph, Guelph, ON, Canada

^eCenter for Computation & Technology, Louisiana State University, Baton Rouge, USA

^fDepartment of Physics & Astronomy, Louisiana State University, Baton Rouge, USA

^gTAPIR, Walter Burke Institute for Theoretical Physics, Mailcode 350-17, California Institute of Technology, Pasadena, CA 91125, USA

^hCanadian Institute for Theoretical Astrophysics, University of Toronto, Toronto M5S 3H8, Canada

ⁱDepartment of Physics, University of Toronto, 60 St. George Street, Toronto, ON M5S 3H8, Canada

Abstract

We introduce a new relativistic astrophysics code, SpECTRE, that combines a discontinuous Galerkin method with a task-based parallelism model. SpECTRE’s goal is to achieve more accurate solutions for challenging relativistic astrophysics problems such as core-collapse supernovae and binary neutron star mergers. The robustness of the discontinuous Galerkin method allows for the use of high-resolution shock capturing methods in regions where (relativistic) shocks are found, while exploiting high-order accuracy in smooth regions. A task-based parallelism model allows efficient use of the largest supercomputers for problems with a heterogeneous workload over disparate spatial and temporal scales. We argue that the locality and algorithmic structure of discontinuous Galerkin methods will exhibit good scalability within a task-based parallelism framework. We demonstrate the code on a wide variety of challenging benchmark problems in (non)-relativistic (magneto)-hydrodynamics. We demonstrate the code’s scalability including its strong scaling on the NCSA Blue Waters supercomputer up to the machine’s full capacity of 22,380 nodes using 671,400 threads.

Keywords: Discontinuous Galerkin, Hydrodynamics, Magnetohydrodynamics, Task-based parallelism

1. Introduction

Numerical simulation of astrophysical phenomena is a computationally challenging task. The relevant equations are often coupled nonlinear partial differential equations (PDEs) with complicated microphysics. High fidelity simulations necessarily require extremely large computational grids in all three spatial dimensions on which non-uniform workloads must be efficiently parallelized. A simulation may involve large spatial and temporal dynamic ranges, and develop (magneto-)hydrodynamic shocks and turbulent flows.

Accurate numerical simulations of astrophysical systems such as neutron star mergers [1–3] and core-collapse supernovae [4–6] are crucial for achieving the full scientific potential of current and future experiments such as the Fermi Gamma-Ray Space Telescope [7] and the Laser Interferometer Gravitational-Wave Observatory [8]. Yet for many of these systems the computational errors are often too large (or not even quantifiable) with current algorithmic and hardware limitations. The simulations also take too long, several weeks to many months of wall time on present supercomputers, precluding explorations of the theoretical parameter space.

Within the astrophysics communities employing grid-based methods, the industry standard has been finite-volume or finite-difference methods parallelized by distributing cells across processors and communicating data with message passing interface (MPI). The evolution is synchronized according to a global simulation time. A

variety of astrophysics codes (e.g., Refs. [6, 9–14]) have been designed based on these fundamental building blocks.

These strategies work well when the computations are reasonably homogeneous or when one seeks good parallelization to only a few thousand cores. As the number of MPI processes increases, so does the cost of communication which, together with non-uniform workload typical of astrophysics problems, limits the maximum number of useful cores that codes can run on. Efficient core utilization becomes non-trivial, often requiring careful optimization by hand to achieve good scalability [15]. Finite-volume and finite-difference methods do not easily parallelize well beyond second-order accuracy because of increasingly large stencil sizes, leading to more communication between cells.

As one looks ahead to the arrival of exascale computing, it will become increasingly important to focus on developing algorithms that can take full advantage of these very large machines.

Discontinuous Galerkin (DG) methods [16–21], together with a task-based parallelization strategy, have the potential to tackle many of these problems. DG methods offer high-order accuracy in smooth regions, robustness for shocks and other discontinuities, and grid flexibility including a formulation that allows for comparatively straightforward *hp*-adaptivity and local timestepping. DG methods are also well suited for parallelization: Their formulation in terms of local, non-overlapping elements requires only nearest-neighbor communication regardless of the scheme’s order of convergence.

Despite extensive success in engineering and applied mathematics communities over the past two decades, applications in relativity [22–25] and astrophysics [26–29] have typically been exploratory or confined to simple problems. Within the past year, however, there have been significant advances toward production codes for non-relativistic [30] and relativistic [31, 32] hydrodynamics, special relativistic magnetohydrodynamics [33], and the Einstein equations [34]. These codes use MPI to implement a data parallelism strategy.

In this paper, we describe SpECTRE, a general purpose discontinuous Galerkin solver for relativistic astrophysics. A distinguishing feature of SpECTRE is its task-based parallelism model. Instead of dividing work between parallel processes based on cell ownership, the algorithm is decomposed into a list of tasks and their inter-dependencies. Examples of tasks include, for example, computing a derivative in an element, computing a numerical flux on a boundary or taking a time step. Tasks are assigned to processes/threads dynamically during the computation, in such a way as to satisfy dependencies and to minimize the number of idle cores. When a core becomes idle, it is given another task to complete. This framework is very different from the more traditional synchronous, data parallelism model used in other grid-based astrophysics codes.

The algorithm’s scalability is achieved through (i) separation of the tasks of communication and computation, so that they can be overlapped, (ii) asynchronous, non-blocking communication so that cores are not idle, and (iii) a runtime system to manage task queues, distribute tasks to cores, and gather timing statistics to inform load-balancing decisions. The power of task-based parallelism has already shown impressive performance in other application domains, for example Refs. [35–41]. SpECTRE uses the Charm++ library [42–45] to implement this parallelism model.

This paper is organized as follows. The hydrodynamic systems currently solved by SpECTRE are summarized in §2, and include the non-relativistic Euler equation and the relativistic (magneto-)hydrodynamics systems in arbitrary gravitational fields. Next, in §3 we describe a nodal DG scheme and those approximate Riemann solvers and high-resolution shock capturing limiters that we implement within SpECTRE. DG schemes naturally map into a task-based parallelism framework, and in §4 we describe how the algorithm can be broken down into tasks and subsequently parallelized using Charm++. Next, we present a sampling of results for standard performance (see §5) and benchmark (see §6) tests. Our scalability experiments demonstrate the power of a task-based approach. A key result is Fig. 5, which shows excellent strong scalability on the Blue Waters machine up to its full capacity of 22,380 nodes using 671,400 threads.

2. Conservation laws

SpECTRE is designed to solve conservation laws written in flux conservative form. A familiar example we shall consider is the non-relativistic (or Newtonian) Euler equation of hydrodynamics. Other conservation laws, such as the relativistic (magneto-)hydrodynamics system, are posed on general spacetimes equipped with

a metric $g_{\mu\nu}$. These systems are most easily discussed with tensor notation. The details of these systems are not necessary to understand the rest of the paper, which avoids the use of tensors.

2.1. Preliminaries and notation

The metric in a general spacetime can be written in the standard space plus time form (See, e.g., [46] or [47]),

$$ds^2 = g_{\mu\nu}dx^\mu dx^\nu = -\alpha^2 dt^2 + \gamma_{ab}(dx^a + \beta^a dt)(dx^b + \beta^b dt). \quad (2.1)$$

Here α is called the lapse function, β^a the shift vector, and γ_{ab} is the spatial metric on $t = \text{constant}$ hypersurfaces, sometimes called time slices. Here and throughout, repeated indices are summed over following Einstein's summation convention. In d spatial dimensions, Greek indices μ, ν, \dots range from 0 to d , Latin indices a, b, \dots will be purely spatial, ranging from 1 to d . In a flat spacetime (no relativistic gravitational field), we can set $\alpha = 1$, $\beta^a = 0$. Furthermore, in Cartesian coordinates, for a flat spacetime the spatial metric is simply the Euclidean metric $\gamma_{ab} = \delta_{ab}$. Throughout this paper, we work in “code units” in which the speed of light $c = 1$ for all the relativistic applications while for gravity applications Newton's constant $G = 1$ and the solar mass $M_\odot = 1$.

A conservation law posed on a generic spacetime takes the form

$$\frac{1}{\sqrt{\gamma}}\partial_t(\sqrt{\gamma}U) + \frac{1}{\sqrt{\gamma}}\partial_a(\sqrt{\gamma}F^a) = S, \quad (2.2)$$

where γ is the determinant of γ_{ab} . We are interested in numerically solving (2.2) as an initial-boundary value problem over the domain Ω subject to appropriate initial and boundary data. Additional geometric factors present in Eq. (2.1) are hidden inside the definitions of the state vector, U , flux vectors, F^a , and source vector S . We denote the length of these vectors by n . We now consider three particular systems that we implement in SpECTRE.

2.1.1. Newtonian Euler hydrodynamics

In d spatial dimensions, the Euler equations form a set of $d + 2$ coupled nonlinear conservation laws. These can be written in the more general form (2.2) (with $\gamma = 1$ in Cartesian coordinates) with an evolved state vector of conserved quantities

$$U = [\rho, \rho v_i, E]^T, \quad (2.3)$$

flux vectors

$$F^a = [\rho v^a, \rho v_i v^a + p\delta_i^a, (E + p)v^a]^T, \quad (2.4)$$

and (for the examples explored in this paper) a vanishing source vector S . Here ρ is the density, E is the energy density, p is the pressure, and $v^i = \gamma^{ij}v_j (= v_i$ in Cartesian coordinates) are the components of the fluid flow velocity. Let ϵ be the internal energy per unit mass. Then the energy can be written as $E = \rho\epsilon + \frac{1}{2}\rho v^i v_i$. The system is completed by an equation of state, which in general can be written as $p = p(\rho, \epsilon)$ and may also depend on the fluid's composition. For the simple equations of state considered in this paper, the relations between conserved and primitive variables (ρ, v_i, ϵ) are simple algebraic expressions.

2.1.2. Relativistic Euler hydrodynamics

In d spatial dimensions, the relativistic Euler equations form a set of $d + 2$ coupled nonlinear conservation laws with the evolved state vector of conserved quantities given by (e.g., [48])

$$\sqrt{\gamma}U = \begin{pmatrix} \tilde{D} \\ \tilde{S}_i \\ \tilde{\tau} \end{pmatrix} = \begin{pmatrix} \sqrt{\gamma}\rho W \\ \sqrt{\gamma}\rho h W^2 v_i \\ \sqrt{\gamma}(\rho h W^2 - p - \rho W) \end{pmatrix}, \quad (2.5)$$

where the components of the fluid's 3-velocity, $v^i = \gamma^{ij}v_j$, are defined from the fluid's 4-velocity, $u^\mu = W(1, v^i)$, $W \equiv \alpha u^0 = 1/\sqrt{1 - v^i v_i}$ is the Lorentz factor, $h = 1 + \epsilon + p/\rho$ is the relativistic specific enthalpy, and ϵ is the

specific internal energy in the fluid's rest frame. The pressure is given by a general equation of state $p(\rho, \epsilon)$. Here a tilde denotes a “densitized” version of the quantity, for example $\tilde{D} = \sqrt{\gamma}D$. Define the fluid “transport velocity” to be $v_{\text{tr}}^i = \alpha v^i - \beta^i$. Then the flux and source vectors are given by

$$\sqrt{\gamma}F^a = \begin{pmatrix} \tilde{D}v_{\text{tr}}^a \\ \tilde{S}_i v_{\text{tr}}^a + \sqrt{\gamma}\alpha p \delta_i^a \\ \tilde{\tau}v_{\text{tr}}^a + \sqrt{\gamma}\alpha p v^a \end{pmatrix}, \quad \sqrt{\gamma}S = \begin{pmatrix} 0 \\ (\alpha/2)\tilde{S}^{lm}\partial_i\gamma_{lm} + \tilde{S}_k\partial_i\beta^k - \tilde{E}\partial_i\alpha \\ \alpha\tilde{S}^{lm}K_{lm} - \tilde{S}^l\partial_l\alpha \end{pmatrix}, \quad (2.6)$$

with the source \tilde{S}^{ij} and the energy \tilde{E} given as

$$\tilde{S}^{ij} = \sqrt{\gamma}\rho h W^2 v^i v^j + \sqrt{\gamma}p\gamma^{ij}, \quad (2.7)$$

$$\tilde{E} = \sqrt{\gamma}\rho h W^2 - \sqrt{\gamma}p, \quad (2.8)$$

and where K_{ab} is the extrinsic curvature tensor of the $t = \text{constant}$ slice and $K = \gamma^{ab}K_{ab}$ its trace. For a flat metric written in Cartesian coordinates ($\alpha = 1$, $\beta^a = 0$, $\gamma_{ab} = \delta_{ab}$, $\gamma = 1$, $K_{ab} = 0$, and $K = 0$) the system simplifies to a set of conservation laws appropriate for special relativistic hydrodynamics and the source vector S is identically zero. We use the method outlined in Appendix C of [49] to convert between conserved and primitive variables (ρ , v_i , ϵ).

2.1.3. Relativistic magnetohydrodynamics

The general relativistic equations of ideal magnetohydrodynamics (GRMHD) can be written in conservative form similar to the equations of hydrodynamics. The main complication is the constraint in Maxwell's equations that the divergence of the magnetic field vanishes. Constraint-satisfying initial data evolved according to Maxwell's equations remain constraint satisfying when the evolution equations are solved analytically. Small constraint violations can, however, be numerically unstable. In order to avoid the growth of unstable modes, we adopt an approach known as divergence cleaning [50] that is used in some finite-difference codes [51–53]. Divergence cleaning introduces a new evolved variable, Φ , that couples to the divergence of the magnetic fields, and damps the unphysical mode in the solution.

The exact form of the evolution equations that we use is a modification of those presented in [53], in which we recast the evolution equation for the divergence-cleaning scalar Φ in conservative form. Define B^μ as the magnetic field measured by an observer moving along the normal to a $t = \text{constant}$ slice (Eulerian observer) and b^μ as the magnetic field measured by an observer comoving with the fluid. The two quantities are related by the equations

$$\alpha b^0 = W B^i v_i, \quad (2.9)$$

$$b_i = \frac{B_i}{W} + v_i \alpha b^0. \quad (2.10)$$

We also define a set of auxiliary variables

$$b^2 = b^\mu b_\mu, \quad (2.11)$$

$$(\rho h)^* = \rho h + b^2, \quad (2.12)$$

$$p^* = p + \frac{b^2}{2}. \quad (2.13)$$

The evolved variables are then

$$\sqrt{\gamma}U = \begin{pmatrix} \tilde{D} \\ \tilde{S}_i \\ \tilde{\tau} \\ \tilde{B}^i \\ \tilde{\Phi} \end{pmatrix} = \sqrt{\gamma} \begin{pmatrix} \rho W \\ (\rho h)^* W^2 v_i - \alpha b^0 b_i \\ (\rho h)^* W^2 - p^* - (\alpha b^0)^2 - \rho W \\ B^i \\ \Phi \end{pmatrix}. \quad (2.14)$$

The flux and source vectors are given by

$$\sqrt{\gamma}F^j = \begin{pmatrix} \tilde{D}v_{tr}^j \\ \tilde{S}_i v_{tr}^j + \sqrt{\gamma}\alpha p^* \delta_i^j - \alpha \tilde{B}^j b_i / W \\ \tilde{\tau} v_{tr}^j + \alpha \sqrt{\gamma} p^* v^j - \alpha^2 b^0 \tilde{B}^j / W \\ v_{tr}^j \tilde{B}^i - \alpha v^i \tilde{B}^j + \alpha \gamma^{ij} \tilde{\Phi} \\ \alpha \tilde{B}^j - \tilde{\Phi} \beta^j \end{pmatrix}, \quad \sqrt{\gamma}S = \begin{pmatrix} 0 \\ (\alpha/2) \tilde{S}^{lm} \partial_l \gamma_{lm} + \tilde{S}_k \partial_i \beta^k - \tilde{E} \partial_i \alpha \\ \alpha \tilde{S}^{lm} K_{lm} - \tilde{S}^l \partial_l \alpha \\ -\tilde{B}^j \partial_j \beta^i + \Phi \partial_j (\alpha \sqrt{\gamma} \gamma^{ij}) \\ \alpha \tilde{B}^k \partial_k \ln \alpha - \alpha K \tilde{\Phi} - \alpha \kappa \tilde{\Phi} \end{pmatrix}, \quad (2.15)$$

with the source \tilde{S}^{ij} and the energy density \tilde{E} given as

$$\tilde{S}^{ij} = \sqrt{\gamma} \left((\rho h)^* W^2 v^i v^j + p^* \gamma^{ij} - \gamma^{ik} \gamma^{jl} b_k b_l \right), \quad (2.16)$$

$$\tilde{E} = \tilde{\tau} + \tilde{D}. \quad (2.17)$$

We solve for the primitive variables using the algorithm proposed by Newman and Hamlin [54]. Note that the exact divergence-free (no-monopole) condition, $\Phi = \partial_i \tilde{B}^i = 0$, is analytically preserved by these equations, while numerically the constraint violating mode will now be damped at a rate κ . The coefficient κ is a damping parameter, which we typically set in the range $\kappa \in [0.1, 1]$. Constraint violating modes propagate at the speed of light.

3. A nodal discontinuous Galerkin method

3.1. The algorithm

Following Refs. [17, 31], this section describes the nodal DG method we have implemented. The algorithm is derived using the following steps (details in Ref. [31]):

- Divide the spatial domain into elements. Each element is a mapping of a reference cube (in 3D) or square (in 2D) with extents $[-1, 1]$ in each direction. The mapping is some time-independent function

$$\mathbf{x} = \mathbf{x}(\bar{\mathbf{x}}) \quad (3.1)$$

with Jacobian matrix

$$\mathbf{J} = \left(\frac{\partial x^a}{\partial \bar{x}^{\bar{a}}} \right) \quad (3.2)$$

and Jacobian

$$J = \det \mathbf{J}. \quad (3.3)$$

Here the barred coordinates are standard Cartesian-like coordinates covering the reference element.

- In each element, each component of the quantities $\sqrt{\gamma}U$, $\sqrt{\gamma}F^a$, and $\sqrt{\gamma}S$ is expanded in polynomial basis functions. We choose these basis functions to be a tensor product of 1D basis functions ℓ_i on the reference element, so that the expansion of a typical variable takes the form

$$U(\bar{\mathbf{x}}) = \sum_{ijk} U_{ijk} \ell_i(\bar{x}^1) \ell_j(\bar{x}^2) \ell_k(\bar{x}^3), \quad (3.4)$$

where the time-dependent coefficients $U_{ijk}(t)$ are found from Eq. 3.10 below and $U_{ijk}(0)$ is given in terms of the initial data (see §3.4). The 1D basis functions of degree N are simply the Lagrange interpolating polynomials corresponding to Legendre polynomials,

$$\ell_i(\bar{x}) = \prod_{\substack{j=0 \\ j \neq i}}^N \frac{\bar{x} - \bar{x}_j}{\bar{x}_i - \bar{x}_j}, \quad (3.5)$$

where \bar{x}_i are the nodes of a Gauss-Legendre-Lobatto (GLL) quadrature. These nodes may be found with standard algorithms, for example Algorithm 24 of Ref. [55].

- In each element, follow the standard DG procedure of integrating Eq. (2.2) multiplied by a basis function over the proper volume $\sqrt{\gamma}d^3x$ of the element, where d^3x is the coordinate volume element $dx dy dz$,

$$\int [\partial_t (\sqrt{\gamma}U) + \partial_a (\sqrt{\gamma}F^a) - \sqrt{\gamma}S] \phi_i(\mathbf{x}) d^3x = 0. \quad (3.6)$$

Use integration by parts (Gauss's Theorem) to convert the divergence term to a surface integral:

$$\begin{aligned} \int \partial_a (\sqrt{\gamma}F^a) \phi_i(\mathbf{x}) d^3x &= \int \partial_a (\sqrt{\gamma}F^a \phi_i(\mathbf{x})) d^3x - \int \sqrt{\gamma}F^a \partial_a \phi_i(\mathbf{x}) d^3x \\ &= \oint F^a n_a \phi_i d^2\Sigma - \int \sqrt{\gamma}F^a \partial_a \phi_i(\mathbf{x}) d^3x. \end{aligned} \quad (3.7)$$

Here $d^2\Sigma$ is the proper surface element of the cell and n_a is the unit outward normal.

With a formulation like (3.7) in each element, there is no connection between the elements. The heart of the DG method is to replace F^a in the surface term by the numerical flux F^{a*} , a function of the state vector on *both* sides of the interface:

$$\int \partial_a (\sqrt{\gamma}F^a) \phi_i(\mathbf{x}) d^3x \rightarrow \oint F^{a*} n_a \phi_i d^2\Sigma - \int \sqrt{\gamma}F^a \partial_a \phi_i(\mathbf{x}) d^3x. \quad (3.8)$$

Now undo the integration by parts on the right-hand side of Eq. (3.8):

$$\int \partial_a (\sqrt{\gamma}F^a) \phi_i(\mathbf{x}) d^3x \rightarrow \oint (F^{a*} - F^a) n_a \phi_i d^2\Sigma + \int \partial_a (\sqrt{\gamma}F^a) \phi_i(\mathbf{x}) d^3x. \quad (3.9)$$

- Evaluate the integrals by using the expansion in basis functions, mapping to the reference element with Eq. 3.1, and GLL quadrature. The final result is Eq. (3.17) of Ref. [31]:

$$\begin{aligned} &\frac{d(\sqrt{\gamma}U)_{ijk}}{dt} + \left[\frac{\partial x^1}{\partial x^a} \Big|_{ijk} \sum_l D_{il}^1 (\sqrt{\gamma}F^a)_{ljk} + \frac{\partial x^2}{\partial x^a} \Big|_{ijk} \sum_m D_{jm}^2 (\sqrt{\gamma}F^a)_{imk} \right. \\ &\quad \left. + \frac{\partial x^3}{\partial x^a} \Big|_{ijk} \sum_n D_{kn}^3 (\sqrt{\gamma}F^a)_{ijn} \right] - (\sqrt{\gamma}S)_{ijk} \\ &= -\frac{1}{w_N} F_{i j N} \frac{\sqrt{(2)}\gamma_{ij}}{J_{ijN}} \delta_{kN} + \frac{1}{w_0} F_{i j 0} \frac{\sqrt{(2)}\gamma_{ij}}{J_{ij0}} \delta_{k0} - \frac{1}{w_N} F_{N j k} \frac{\sqrt{(2)}\gamma_{jk}}{J_{N j k}} \delta_{iN} \\ &\quad + \frac{1}{w_0} F_{0 j k} \frac{\sqrt{(2)}\gamma_{jk}}{J_{0 j k}} \delta_{i0} - \frac{1}{w_N} F_{i N k} \frac{\sqrt{(2)}\gamma_{ik}}{J_{i N k}} \delta_{jN} + \frac{1}{w_0} F_{i 0 k} \frac{\sqrt{(2)}\gamma_{ik}}{J_{i 0 k}} \delta_{j0}. \end{aligned} \quad (3.10)$$

Here D_{il}^1 is the differentiation matrix

$$D_{il}^1 = \partial_1 \ell_l(x^1) \Big|_i \quad (3.11)$$

for x^1 , and similarly for the 2- and 3-coordinates. The quantity F is the normal component of the flux difference, $F = (F^{a*} - F^a)n_a$, where F^{a*} is the numerical flux and n_a are the components of the element's locally outward-pointing normal vector (see Fig. 1). The quantity $^{(2)}\gamma$ is the determinant of the 2-dimensional metric induced on the surface by γ_{ij} , and w_0 and w_N are the weights of the Gauss-Lobatto quadrature at the endpoints of the interval.

Equation (3.10) is integrated in time with a suitable ODE time stepper.¹

¹There is an important technical issue that arises when implementing Eq. (3.10) in a code like SpECTRE. In Eq. (3.18) of Ref. [31] it was shown that the boundary flux terms on the right-hand side of Eq. (3.10) can be simplified by using the *unnormalized* normal vector when computing the fluxes. However, we do not use this simplification and instead use the *unit* normal n_a explicitly. The reason is that when the metric or mapping terms differ on the two sides of the boundary, as they will in curved spacetimes or with grid refinement, it is the unit normal that is the same on the two sides of the boundary, whereas pieces of the normalization factor like $\sqrt{(2)}\gamma$ and J differ.

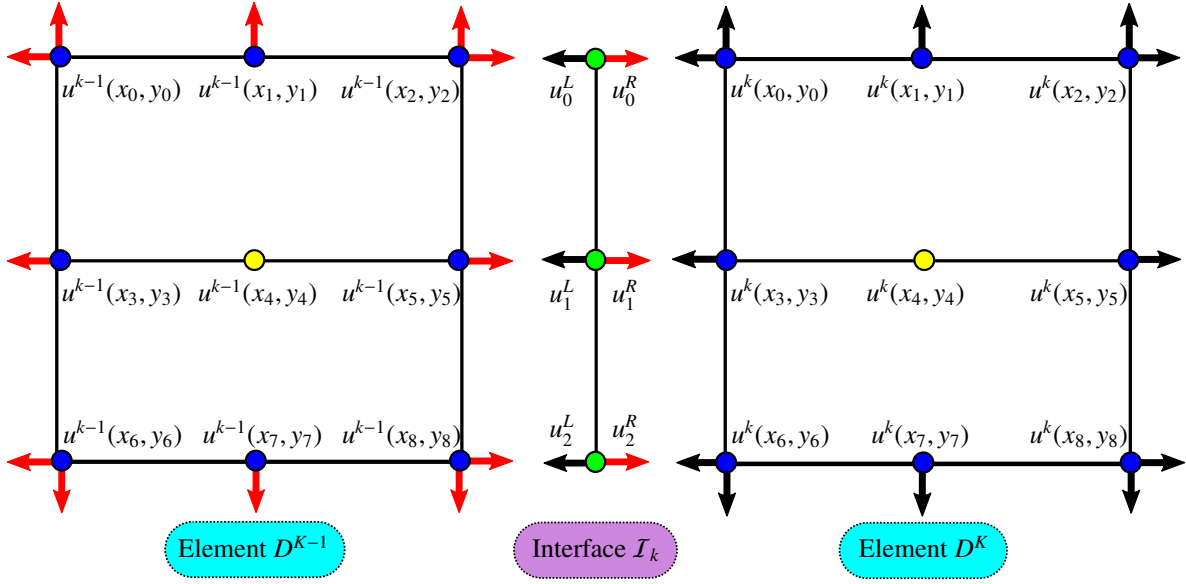


Figure 1: In two spatial dimensions the DG computational domain consists of a collection of elements (rectangles) with touching surfaces (lines) which we call interfaces. On each element the local numerical expansion gives function values at the set of GLL nodal points, shown here with second order elements. While nodal points along each interface can be identical, for example the upper-right point in D^{k-1} is the same as the upper-left point in D^k , the grid values may not be, $u_0^L = u^{k-1}(x_2, y_2) \neq u^k(x_0, y_0) = u_0^R$. Numerical fluxes are computed on each interface using grid data from two neighboring elements (blue circles) and local outward-pointing normals (red and black arrows). Our implementation of the DG scheme treats each interface and element as a C++ object equipped with tasks (see Fig. 2).

Note that in the derivation of Eq. (3.10), each product of expansions is evaluated using a single expansion with coefficients equal to the product of the original coefficients. This replacement leads to an aliasing error: contributions from the high order polynomials are aliased back onto the basis. While this does not affect the precision of the scheme, it can lead to an aliasing-driven instability, which may need to be dealt with by filtering [17].

In this paper, we will consider only the affine mapping from the reference cube for each element:

$$\mathbf{x}(\bar{\mathbf{x}}) = \mathbf{a}^k + \frac{1}{2}(1 + \bar{\mathbf{x}})(\mathbf{b}^k - \mathbf{a}^k), \quad (3.12)$$

where \mathbf{a}^k and \mathbf{b}^k are the lower-left and upper-right coordinates of the element. However, the code is designed to handle arbitrary mappings.

3.2. Numerical fluxes

To complete the DG scheme we must specify functional forms for the components of the numerical flux introduced in the previous subsection. The numerical flux is determined by functions $F^{a*}(U^L, U^R)$ where U^L and U^R are respectively the left and right (or lower and upper) boundary values of the numerical approximation defined on neighboring elements (see Fig. 1). To build a stable, convergent DG scheme the numerical flux must satisfy a few basic properties [17] such as consistency $F^{a*}(U, U) = F^a(U, U)$. For smooth solutions, simple upwinding is a good prescription for the numerical flux. For non-smooth solutions, the flux is typically prescribed by an approximate Riemann solver borrowed from the finite-volume community. In both cases, the numerical flux will, to varying degrees, make use of the characteristic decomposition of the conservation laws (2.2). In non-conservative form, this system of equations can be written as

$$\partial_t U + A^a(U) \partial_a U + \dots = 0, \quad (3.13)$$

where $A^a(U) = \partial_U F^a(U)$ is the flux Jacobian and the neglected terms contain no derivatives of U . Since we assume that the conservation law (2.2) gives a well-posed initial-boundary value problem, it follows that for any covector n_a the matrix

$$n_a A^a(U) \equiv A(U; n) = K(U; n) \Lambda(U; n) K^{-1}(U; n) \quad (3.14)$$

is diagonalizable. Here, $K(U; n)$ is an n -by- n matrix whose i^{th} column is the right eigenvector of $A(U; n)$ corresponding to the eigenvalue λ_i . The eigenvalues (i.e. wavespeeds) are real, ordered

$$\lambda_1(U; n) \leq \lambda_2(U; n) \leq \dots \leq \lambda_n(U; n),$$

and comprise the entries of the diagonal matrix $\Lambda(U; n) = \text{diag}(\lambda_1, \dots, \lambda_n)$. For the (relativistic) hydrodynamics systems considered here, the characteristic decompositions can be found in standard references (e.g. [56, 57]). For the GRMHD system we are unaware of analytical expressions for the full eigensystem, but the speeds can be computed as the roots of polynomials given, e.g., in Ref. [58].

If you apply the DG algorithm of §3.1 to Eq. (3.13), you find that the analytic boundary flux is

$$n_a F^a = n_a A^a U. \quad (3.15)$$

The numerical flux is chosen to emulate this behavior, with freedom to use varying amounts of information from the characteristic decomposition.

We now summarize the numerical fluxes that we have implemented in our code. Later, numerical experiments will provide some insight into the strengths and weaknesses of each choice.

3.2.1. Local Lax-Friedrichs

Because of its simplicity, robustness, and low cost-of-evaluation, the local Lax-Friedrichs (LLF) numerical flux is a popular choice. However, it also introduces a large amount of dissipation into the scheme for low polynomial orders and does not use the system's wave structure, which forms the building blocks of more sophisticated approximate Riemann solvers. At each interface, the LLF flux is computed as

$$(F^{a*} n_a)^{\text{LLF}} = \frac{F^a(U^L) n_a + F^a(U^R) n_a}{2} - \frac{\alpha^{\text{LF}}}{2} (U^R - U^L), \quad (3.16)$$

where α^{LF} is the maximum eigenvalue of either $|\Lambda(U^R; n^R)|$ or $|\Lambda(U^L; n^L)|$ computed at each collocation point along the interface. The LLF flux has a simple interpretation in terms of the average of the physical flux plus a dissipative part proportional to α^{LF} . Numerical dissipation is necessary to stabilize the scheme, and setting $\alpha^{\text{LF}} = 0$ is likely to be unstable, for example.

3.2.2. Harten, Lax and van Leer

The numerical flux proposed by Harten, Lax, and van Leer (HLL) [59, 60] is

$$(F^{a*} n_a)^{\text{HLL}} = \frac{c_{\max} F^a(U^L) n_a - c_{\min} F^a(U^R) n_a}{c_{\max} - c_{\min}} + \frac{c_{\min} c_{\max}}{c_{\max} - c_{\min}} (U^R - U^L), \quad (3.17)$$

where c_{\min} and c_{\max} are estimates on the minimum and maximum signal velocities bounding the left-moving and right-moving wavespeeds that arise when solving the Riemann problem. We use the simple, direct estimates proposed by Davis [61]

$$c_{\min} = \min(\lambda_1(U^L), \lambda_1(U^R), 0), \quad c_{\max} = \max(\lambda_n(U^L), \lambda_n(U^R), 0). \quad (3.18)$$

Alternative estimates may perform better for some problems [60]. Similar to the LLF flux, the last term in Eq. (3.17) provides a stabilizing diffusive term (note that $c_{\min} \leq 0$ and $c_{\max} \geq 0$) while the first term is seen to be a weighted average. If $c_{\min} = 0$ or $c_{\max} = 0$, which will occur whenever all the characteristics are moving in the same direction, the HLL flux reduces to pure upwinding.

3.2.3. Roe

Roe's approach [62, 63] replaces the exact problem (3.13) by a linearized approximation

$$\partial_t U + A_{\text{Roe}}^a \partial_a U = 0, \quad (3.19)$$

where the approximate flux Jacobian matrix, $A_{\text{Roe}}^a(U^L, U^R)$, is required to satisfy conditions enforcing consistency, hyperbolicity, and conservation [60]. If such a matrix A_{Roe}^a can be found, the Riemann problem (3.19) can be exactly solved, and the numerical flux is given by

$$(F^{a*} n_a)^{\text{Roe}} = \frac{F^a(U^L) n_a + F^a(U^R) n_a}{2} - \frac{1}{2} \sum_{i=1}^n \alpha_i |\lambda_i^{\text{Roe}}| e_i^{\text{Roe}}. \quad (3.20)$$

Here λ_i^{Roe} and e_i^{Roe} are the i^{th} eigenvalue and right eigenvector of the Roe matrix, and the wave strengths

$$\alpha_i = (E_i^{\text{Roe}})^T (U^R - U^L),$$

are computed from the projection of the state vector's jump onto the normalized left eigenvectors, E_i .

For a non-relativistic ideal gas described by the Euler equations, a key result is that a Roe matrix can be constructed from so-called Roe-averages,

$$\begin{aligned} \hat{u} &= \frac{\sqrt{\rho_L} u_L + \sqrt{\rho_R} u_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}, & \hat{v} &= \frac{\sqrt{\rho_L} v_L + \sqrt{\rho_R} v_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}, & \hat{w} &= \frac{\sqrt{\rho_L} w_L + \sqrt{\rho_R} w_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}, \\ \hat{h} &= \frac{\sqrt{\rho_L} h_L + \sqrt{\rho_R} h_R}{\sqrt{\rho_L} + \sqrt{\rho_R}}, & \hat{\rho} &= \sqrt{\rho_L \rho_R}, \end{aligned} \quad (3.21)$$

through direct evaluation of the analytical flux Jacobian at the Roe-averaged state

$$A_{\text{Roe}} = n_a A^a(\hat{u}, \hat{v}, \hat{w}, \hat{h}). \quad (3.22)$$

These expression have been written in the standard way using $(u, v, w) \equiv (v^1, v^2, v^3)$ as the fluid flow velocity measured in a Cartesian coordinate system. It is easy to check that the matrix (3.22) satisfies the three Roe conditions. For generic equations of state, a Roe matrix can be constructed, but it will *not* be given by simple evaluation at a Roe-averaged state [56, 64]. Furthermore, for the relativistic fluid equations we are unaware of any known Roe matrix construction that satisfies all three Roe conditions. Nevertheless, we follow the accepted practice of defining a Roe-like matrix along the lines just described for a non-relativistic ideal gas. We expect such matrices to satisfy the consistency and hyperbolicity conditions, but they are unlikely to satisfy all of the Rankine-Hugoniot conditions.

3.2.4. Marquina

The flux proposed by Donat and Marquina [65] also makes use of the characteristic decomposition of the problem but, as opposed to the Roe flux, does not require the computation of an intermediate state. Instead, it uses the characteristic right eigenvectors $e_i^{R,L}$, normalized left eigenvectors $E_i^{R,L}$ and eigenvalues $\lambda_i^{R,L}$ of the linearized Jacobian matrices $A_{R,L}^a(U^{R,L})$ evaluated separately for the left and right state vectors $U^{R,L}$. From these variables, we can define the projected variables and fluxes

$$\omega_i^{R,L} = (E_i^{R,L})^T U^{R,L}, \quad \phi_i^{R,L} = (E_i^{R,L})^T F^a(U^{R,L}) n_a. \quad (3.23)$$

The Marquina numerical flux is then defined as

$$(F^{a*} n_a)^{\text{Marquina}} = \sum_{i=1}^n (\phi_i^+ e_L^i + \phi_i^- e_R^i). \quad (3.24)$$

For the i^{th} eigenvector, the fluxes ϕ_i^\pm are defined so that the scheme is upwind if $\lambda_i^R \lambda_i^L \geq 0$, but switches to a more dissipative LLF scheme otherwise. That is, if $\lambda_i^{R,L} \geq 0$, we set $(\phi_i^+ = \phi_i^L; \phi_i^- = 0)$. If $\lambda_i^{R,L} \leq 0$, we choose $(\phi_i^- = \phi_i^R; \phi_i^+ = 0)$. Otherwise, the LLF flux is obtained using

$$\alpha_i = \max(|\lambda_i^R|, |\lambda_i^L|), \quad \phi_i^+ = \frac{1}{2}(\phi_i^L + \alpha_i \omega_i^L), \quad \phi_i^- = \frac{1}{2}(\phi_i^R - \alpha_i \omega_i^R). \quad (3.25)$$

While the Marquina flux has the advantage of avoiding the intermediate state used by the Roe flux, it does require the computation of the eigenvectors and eigenvalues of the characteristic matrix for both the left and right state at each interface. For the relativistic hydrodynamics system, expressions for both sets of eigenvectors can be found, e.g., in Refs. [48, 66].

3.2.5. Numerical dissipation and maximum wave speeds for the relativistic MHD system

As we have just seen, Riemann solvers which do not make use of the characteristic eigenvectors of the problem (e.g., the LLF and HLL numerical fluxes) introduce numerical dissipation proportional to the maximum characteristic speed of the system. For the relativistic MHD system described in Sec. 2.1.3, we do not have analytical expressions for the characteristic eigenvectors and characteristic speeds of the system. We do know, however, that the use of a divergence cleaning method introduces the speed of light as one of the characteristic speeds.

Using the speed of light as the maximum speed in the HLL/LLF fluxes would introduce a large amount of dissipation, while using a smaller characteristic speed in the dissipative component of those fluxes is unstable. Yet, it is easy to see that the fluxes F^l for the divergence cleaning variable $\tilde{\Phi}$ and the longitudinal component of the magnetic field \tilde{B}^l are functions of \tilde{B}^l and $\tilde{\Phi}$, but not of the other evolved variables of the GRMHD equations. Accordingly, at each interface, we can separate the GRMHD equations into two sub-systems: a system of 2 equations for the divergence cleaning variable and the longitudinal component of the field (component normal to the interface), and a system of 7 equations for all other variables. The characteristic eigenvalues for the small system $(\tilde{B}^l, \tilde{\Phi})$ are $\lambda_\pm = -\beta^l \pm \alpha \sqrt{\gamma^l}$, and the corresponding eigenvectors are $(\pm \sqrt{\gamma^l}, 1)$. This sub-system can be treated separately from the larger sub-system of 7 equations. In particular, we can use the HLL/LLF fluxes with the characteristic speeds λ_\pm , but we could also use the Roe or Marquina fluxes (for $\beta^l = 0$ and α continuous across the interface, the Marquina and HLL fluxes are in fact identical in this case).

The larger sub-system of 7 equations has its own complete set of 7 eigenvectors and eigenvalues, and its eigenvectors are orthogonal to those of the smaller sub-system $(\tilde{B}^l, \tilde{\Phi})$. The system also has the same eigenvalues and eigenvectors as the GRMHD equations evolved without divergence cleaning. While we do not know the maximum eigenvalues of that system analytically, we can either solve for them numerically or make use of the known bound on the value of the characteristic speeds in the fluid frame [58]

$$\lambda_{\text{fl}}^2 \leq (v_A^2 + c_s^2(1 - v_A^2)), \quad (3.26)$$

with c_s the relativistic sound speed and v_A the *Alfven* speed.

In the tests presented here, we use the bound on λ_{fl}^2 to determine an approximate minimum and maximum characteristic speed in the grid frame. We use the same method for the computation of the fluxes for both subsystems (either HLL or LLF), but with a different choice of maximum characteristic speed for the sub-system $(\tilde{B}^l, \tilde{\Phi})$, thus recovering stability without adding dissipation to the evolution of the other variables.

3.3. High-resolution shock capturing limiters

It is well known that an approximation of a non-smooth function by smooth polynomials will produce spurious (Gibbs) oscillations with overshoots. In these troublesome elements the DG scheme's order of convergence will be reduced and the solution may differ qualitatively from the true solution. For example, if negative densities are generated from undershoots, the wavespeeds can become imaginary, triggering an instability.

The main challenge is to modify the numerical solution in such a way that the spurious oscillations are removed (or at least acceptably controlled) while retaining as much accuracy as possible. This is the job of *limiters*. As an extreme example, the modification rule could be to set the solution equal to its cell average,

equivalent to retaining only the constant part of the polynomial basis. Such a heavy-handed limiter is guaranteed to remove spurious oscillations but at the price of reducing the scheme's accuracy to first order. More sophisticated limiting procedures must be able to discriminate between spurious oscillations generated near non-smooth features and genuine maxima or minima of the true solution. For nonlinear hyperbolic partial differential equations, an additional complication arises as the solution may develop physical discontinuities. A robust limiter should be able to handle a diverse range of problems without the need for fine-tuned numerical parameters.

Given the importance of the problem, extensive work has gone into the development of high-resolution shock capturing (HRSC) limiters. As the name suggests, HRSC limiters seek to retain the accuracy afforded by the polynomial basis while resolving sharp features such as shocks. They typically involve two conceptually distinct parts. First, a *troubled-cell indicator* is used to identify elements that may contain spurious oscillations. Next, a *limiter* modifies the solution to reduce or completely remove the offending oscillation. When implemented within the Method of Lines framework, this procedure is applied to both the advanced solution and any intermediate stages required by the timestepper.

We now summarize those HRSC limiters that we have implemented. These limiters can all be applied straightforwardly to either the set of conserved, primitive or characteristic variables; in the literature one can find endorsements for all three (e.g. [17, 19, 30, 67]). Unless noted otherwise, our discussion and application of limiters will be applied to the conserved variables.

3.3.1. Slope limiters

Slope limiters were originally introduced in finite-volume methods and were some of the earliest attempts at doing better than setting the troubled cell to its average. These limiters are generally *at best* second-order accurate and work by (i) modifying the solution's slope and (ii) dropping any higher-order terms originally present in the approximate solution (3.4). We will consider a family of minmod-based limiters [17, 19, 68] that replace the numerical solution in element D^k by a piecewise linear representation:

$$U^k \rightarrow U_0^k + a_x(x - x_0) + a_y(y - y_0) + a_z(z - z_0). \quad (3.27)$$

Here a_x, a_y, a_z are estimates of the solution's slope along the three coordinate directions, U_0^k is the solution's mean value, and (x_0, y_0, z_0) is the coordinate center of the element.

We build Eq. (3.27) using data from D^k and its six neighbors by considering each dimension separately. Consider the task of finding a_x . From D^k we use the numerical solution's slope, U_x^k , and cell average, U_0^k . From its left and right neighbors, D^{k-1} and D^{k+1} , we require the numerical solution's cell averages, U_0^{k-1} and U_0^{k+1} . Next, we compute three estimates of the slope in the x -direction:

$$a_1 = U_x^k, \quad a_2 = \frac{U_0^{k+1} - U_0^k}{\Delta x/\beta}, \quad a_3 = \frac{U_0^k - U_0^{k-1}}{\Delta x/\beta}, \quad (3.28)$$

where Δx is the element's length and $\beta = 2$ for standard minmod [17, 69] and $\beta = 1$ for the more dissipative MUSCL (Monotone Upstream Centered Scheme for Conservation Laws) limiter of van Leer [17, 69–71]. If the solution were known to be smooth enough within D^k , we would hope to retain $a_x = U_x^k$ as the slope since it would be the most accurate. For smooth solutions and not-too-large values of Δx , the values a_2 and a_3 are expected to be reasonable estimates of the true slope. Spurious oscillations (and, unfortunately, local maxima and minima) manifest themselves as inconsistent signs of the three estimates. These considerations motivate the introduction of a *minmodB* function

$$a_x = m_B(a_1, a_2, a_3; M) \equiv \begin{cases} a_1, & |a_1| \leq M(\Delta x)^2, \\ m(a_1, a_2, a_3), & \text{otherwise.} \end{cases} \quad (3.29)$$

where the *minmod* function is

$$m(a_1, a_2, a_3) = \begin{cases} \text{sign}(a_1) \min(|a_1|, |a_2|, |a_3|), & \text{sign}(a_1) = \text{sign}(a_2) = \text{sign}(a_3), \\ 0, & \text{otherwise.} \end{cases} \quad (3.30)$$

The minmodB function m_B returns a value that we take to be the estimated slope a_x . The positive constant M controls the amount of oscillation we are willing to tolerate. If $M = 0$, the slope limiting procedure guarantees that the numerical solution will be total variation diminishing in the mean (i.e., no spurious oscillations in the cell averages), but necessarily clips off the solution's extrema. When $M \neq 0$ we allow local maxima and minima to potentially escape limiting, but continue to suppress large, spurious oscillations. Ideally M would be chosen proportional to the solution's second derivative at the extrema, although in practice its value may not require any special fine-tuning [68]. Estimated values for the slopes a_y and a_z follow the exact same recipe.

To decide whether the solution is in need of limiting, we (i) project the solution onto the space of piecewise linear functions and then (ii) modify the linearized solution's slope using the minmodB function. Notice that the minmodB slope limiter acts as both a troubled cell detector and limiter; for a piecewise linear basis no limiting occurs whenever $a_1 = m_B(a_1, a_2, a_3; M)$. For a piecewise linear basis this is reasonable, whereas for higher-order bases we use a generalization described next.

3.3.2. Generalized high-order minmodB limiter

Whenever our basis functions are of degree $N \geq 2$, we use a high-order generalization of the minmodB limiter as a troubled cell detector. Following Ref. [19], we (i) linearize

$$U^{\text{Lin}} = U_0^k + U_x^k(x - x_0) + U_y^k(y - y_0) + U_z^k(z - z_0), \quad (3.31)$$

by dropping all of the higher-order terms originally present in the numerical solution (3.4), (ii) apply the minmodB slope limiter to U^{Lin} and finally (iii) check if the minmodB limiter has modified U^{Lin} . If U^{Lin} was modified, we replace the numerical solution with the minmodB limited solution computed in step (ii). Otherwise, the full high-degree polynomial solution is used and no limiting is performed.

3.3.3. Limiter of Moe, Rossmanith and Seal

Moment limiters extend the idea of modifying the solution's slope information to their higher-moment analogues, like those terms in the approximation (3.4) proportional to x^3 , xyz , yz^2 etc. We will consider a recently proposed moment limiter of Moe, Rossmanith and Seal (MRS) [67] which has the following desirable properties: in smooth regions it retains the full high-order accuracy afforded by the basis, its implementation requires minimal communication between neighbors, and it has demonstrated good performance on a variety of benchmark tests [67].

Instead of the ansatz (3.27), we now seek to replace the numerical solution as follows:

$$U^k \rightarrow U_0^k + \Theta_k (U^k - U_0^k). \quad (3.32)$$

This replacement rescales all of the higher-order moments by the rescaling function $\Theta_k \in [0, 1]$. In the presence of large spurious oscillations we expect $\Theta_k = 0$, and the numerical solution is replaced by its cell average. In smooth regions we expect $\Theta_k = 1$, and the solution remains unaltered. To implement the MRS limiter, we follow the steps described in Ref. [67] and, in particular, for systems of equations we use the primitive variables to find Θ_k whereas the rescaling (3.32) is carried out on the conserved variables, thereby retaining conservation.

The effectiveness of this limiter depends on a free parameter, $\tilde{\alpha} \geq 0$, which is used to estimate the solution's upper and lower bound on each element. Following Ref. [67], we compute these estimates from neighbor data and the value of $U_0^k \pm \tilde{\alpha} h^{3/2}$ (we take $\alpha(h) = \tilde{\alpha} h^{3/2}$ in Eqs. 3.1 and 3.2 of Ref. [67]). Here $h = \Delta x$ is the width of a uniform element and the bounds are computed separately for each component of the vector of primitives. A non-zero value of $\tilde{\alpha}$ is required to retain high-order accuracy near smooth extrema while $\tilde{\alpha} = 0$ suppresses both extrema and spurious oscillations. In practice we find a wide range of values are effective for a given problem. As shown in Ref. [67], for any non-zero value of $\tilde{\alpha}$ the MRS limiter will turn off around smooth extrema as the grid is refined $h \rightarrow 0$.

3.4. Initial data, boundary conditions and timesteppers

Spatial approximation of the underlying system by the DG strategy leads to Eq. (3.10). Time integration of this equation can then be carried out with a suitable ODE integrator, that is, we are using the method-of-lines. We

have implemented two third-order integrators: a multi-step Adams-Bashforth (AB3) and a multi-stage strong stability-preserving Runge-Kutta timestepper [72, 73]. A CFL condition restricts the largest stable timestep Δt_{\max} associated with explicit numerical integration of Eq. (3.10). For a DG scheme, it is known that Δt_{\max} is inversely proportional to the largest wavespeed, λ_{\max} , arising from the characteristic decomposition (3.14) and proportional to the smallest distance between neighboring GLL points on the physical grid, Δx_{\min} . Since the value of Δt_{\max} in fact depends on all aspects of the scheme, such as details of the numerical flux and the timestepper’s stability region, we find a stable Δt from the scaling relation

$$\Delta t_{\max} \sim C \frac{\Delta x_{\min}}{\lambda_{\max}},$$

and a reasonable estimate for the unknown scaling factor C .

Initial data for Eq. (3.10) is provided by interpolating onto the numerical grid. At the interface located on the domain’s physical outer boundary, we use the same numerical flux used for interior interfaces but now supply the exterior solution. Throughout this paper we use periodic or analytic physical boundary conditions for the exterior state.

3.5. Computational considerations

The time-to-solution is primarily determined by three different cost factors: computation, data movement, and inter-process communication.

Computation is probably the most well known of these. Computational cost is often measured in Flop (floating-point operations), and speed is measured in Flop/s (floating-point operations per second). A high-performance compute node can today execute about 1000 GFlop/s. Of course, such a compute speed can only be obtained if there are no other bottlenecks in the code, and usually there are.

Data movement is today for many algorithms the actual limiting factor. A core’s registers and caches can hold only a small amount of data, and accessing the main memory is slow, both in the sense of large latencies and small bandwidths, when compared with the theoretical computational peak performance of a node. A typical latency for loading data from memory is of the order of 100 ns, and a typical bandwidth is of the order of 10 GByte/s to 100 GByte/s. This memory bandwidth is shared between all the cores of a socket.

The relevant property of a computational algorithm is its *byte-per-Flop* ratio, i.e., the average (amortized) number of bytes that need to be loaded from or stored to memory for each floating-point operation. Many algorithms have byte-per-flop ratios of about 1 Byte/Flop, whereas many computing systems have byte-per-flop ratios of only about 0.1 Byte/Flop. In other words, performance (even under the idealized assumptions) is often limited by the available memory bandwidth, and not by the available compute power.

Instead of letting the excess compute power go to waste, one can use it to perform additional computations on data already loaded from memory. For example, when limited by data movement costs, the additional accuracy of a high-order DG algorithm almost comes for free.

Inter-process communication arises from the need to communicate between different processes running on different nodes. Today, all large HPC systems employ a distributed memory architecture requiring inter-process communication. A modern HPC system (using, for example, an InfiniBand network) has a bandwidth on the order of 1 GByte/s to 10 GByte/s and latencies of $1 \mu s$ to $10 \mu s$. The Blue Waters’ Gemini interconnect, which we use in our scalability experiments, achieves a peak bandwidth of 9.6 GByte/s.

In a spatial domain decomposition with a homogeneous workload, the amount of work per element scales with the volume of the element, while the amount of data that needs to be communicated scales with its surface area. The communication cost of an algorithm depends on the “thickness” of the surface layer that needs to be communicated. By construction, the DG algorithm only requires communication of interface data (a thickness of one) regardless of the scheme’s order of convergence (see Figure 1). For example, suppose a 3D domain has N elements in total. When distributed over P processes, the computation and communication cost scale as $O(N/P)$ and $O((N/P)^{2/3})$, respectively. The communication-to-computation ratio is then $O((P/N)^{1/3})$. This limits the number of processes over which a problem with a given size can be distributed; once the communication cost is a significant fraction of the overall cost, increasing P will increase the total cost.

Network latency also contributes to the communication cost. The latency can be ignored if the communication time is much larger than the latency. This is the case if the amount of communicated data is larger

than the network’s *latency-bandwidth-product*. Using the numbers quoted above, current HPC networks have a latency-bandwidth-product of the order of 10 kByte. If typical messages are shorter than this, then one way to hide the latency is to employ a multi-threading system where multiple tasks can initiate independent communications simultaneously, so that some tasks can continue doing useful work while other tasks are blocked on communication.

DG methods are particularly well-suited for parallelization. Their formulation in terms of local, non-overlapping elements requires only nearest-neighbor communication of surface data terms. They achieve higher-order accuracy (which increases the compute cost per element) without increasing the communication cost, unlike, e.g., finite-difference methods where the stencil size grows with the order of accuracy.

4. Task-based parallelism

4.1. General considerations

The de facto standard for parallel and distributed programming to solve large systems of PDEs is based on MPI and multi-threading, usually implemented using OpenMP. The simulation domain is split onto MPI processes and data are communicated across the element boundaries. This approach has the advantage that it is rather straightforward to implement, and it scales well if the computations are reasonably homogeneous. The hallmarks of this approach are globally synchronous phases, alternating between computation and communication.

If different regions of the domain require different calculations, then the resulting *load imbalance* can be difficult to address. In addition, the global communication phases lead to delays that are in principle avoidable, but in practice it is often difficult to overlap computation and communication to hide their impact.

Task-based parallelism approaches distributed computing “from the other end.” Instead of splitting the domain into elements, as few as possible and, each as large as possible (to reduce communication overhead), one splits the computation into many tasks of a certain minimum size. Kaiser et al. [74] and Sterling et al. [75] discuss this in greater detail in the context of programming models for exascale computing, comparing in particular MPI and a task-based parallelism that is similar to the one used here. The advantages of task-based parallelism are clear — for example load balancing is much simplified, as the tasks can simply be moved between processes. However, there is a cost to pay.

Using many small tasks introduces a communication latency delay for each task, as it has to wait for its input data to be communicated to the current process. To remedy this, one employs a *runtime system* (RTS) that keeps track of which task is waiting for what communication, and that executes tasks as their inputs become available, preferably in parallel on multiple threads. This scheme automatically overlaps communication and computation, leading to a much improved overall efficiency. It is obviously non-trivial to develop such a runtime system that executes efficiently on a wide range of HPC architectures. We thus settled on an existing, proven software framework (Charm++ [42]; see below).

A distinguishing feature of SpECTRE is thus its task-based parallelism strategy that naturally avoids the bottlenecks of globally synchronous communication.

We are particularly interested in achieving efficiency and scalability to large ($\gg O(100\text{ k})$) core counts. We expect that the defining features enabling efficient scalability to be (i) separation of the tasks of communication and computation, so that they can overlap, (ii) asynchronous, non-blocking communication so that cores are not idle, and (iii) a runtime system to manage task queues, distribute tasks to cores, and collect timing statistics to inform dynamical load-balancing decisions. We discuss this in more detail in the next subsection.

When choosing the size of the individual tasks, one needs to strike a balance between having as many (small) tasks as possible and making the tasks sufficiently large to overcome the overhead. The relevant overhead here is the time necessary to create a task, examine its dependencies, schedule its execution, and move data in and out of CPU caches. On a modern HPC system, this overhead is likely to be in the range of $1\mu\text{s}$ to $10\mu\text{s}$; consequently, a task should run on average for at least that long to be efficient.

4.2. The Charm++ library

SpECTRE uses the Charm++ library [42–45] to implement a task-based parallelism model. Among the available task-based libraries we explored, in our opinion Charm++ currently provides the best combination of robust performance, a diverse set of features, good documentation, and a large user base.

The building blocks of a Charm++ application are distributed objects called *chares*. Chares are ordinary C++ objects with a few special properties: they inherit from a base class whose code is generated by Charm++ at compile time, they support creating new instances on remote nodes and they support a unique kind of member function called an *entry method*. When an entry method is invoked a new task is created. Charm++ takes care of chare distribution and task routing across the set of available nodes and cores.

SpECTRE’s computation is decomposed into a collection of chares that interact by sending messages to one another. Two interacting chares may reside on different processors, and so the sender cannot directly call the receiver’s entry method. Instead, the sender alerts the Charm++ *runtime system* (RTS) that a new task needs to be processed. This request, along with any relevant data, is packaged into a single message and shipped off to the node on which the receiving chare resides. The message is captured by the RTS running on the receiver’s node and placed into a task-pool. The message remains in the task-pool until the necessary computing resources become available, at which point the receiver’s entry method is called and work on the task begins. A numerical evolution proceeds by asynchronous message-driven execution [76]: executing tasks create new tasks to perform, and so on until there are no more tasks to complete.

The Charm++ RTS continuously monitors the availability of resources and keeps track of a list of pending tasks in a task-pool. Since each instantiated chare resides on a unique node, local task-pools are maintained on each node. When a core becomes available, the runtime system decides which message from the task-pool to process and invokes the entry method specified within the message. Control switches back and forth between the RTS’s scheduler and SpECTRE’s code. A message-driven parallel programming model naturally promotes latency tolerance since there is never an explicit “receiving end” waiting for data to arrive. Instead, the chare remains dormant until all of the necessary remote data has been received. Meanwhile, other tasks may continue to execute on the available computing resources. In particular, overlap of communication and computation is much easier to implement than within a traditional MPI framework. Charm++ also contains features to aid scheduling tasks on GPUs.

In addition to managing task-pools and message routing between physically separated chares, the RTS includes numerous features to improve code performance and maximize processor utilization. For example, the RTS can optimize communications between chares on the same node by exploiting the available shared memory environment. The RTS also records information about computation timing statistics and communication patterns for measurement-based load balancing. For example, certain load balancing strategies may attempt to cluster frequently communicating chares so that they reside on nearby physical locations, or dynamically migrate chares to balance the overall workload [77, 78]. We have not yet explored Charm++’s load balancing features.

4.3. Decomposition of the DG scheme into tasks

The DG algorithm lends itself quite naturally to task-based parallelization. The computational domain provides a natural decomposable unit; each element and interface is represented as a unique object in an *array of chares*. Figure 2 depicts the mapping between a computational grid, the collection of Charm++ chares needed to represent the computational grid, and one possible distribution of these objects onto a set of three cores with one Charm++ process running per core. In this particular example, five element chares and four interface chares are created, each of which contains a portion of the grid data illustrated by Fig. 1. Element and interface chares each define a set of tasks (i.e., entry methods) that they will need to perform.

Each interface chare defines a “Compute Interface Flux” task that is invoked by the upper and lower abutting element chares. Once this task has been called by each neighbor, the boundary term (right-hand side of Eq. 3.10) is computed and each element’s “Advance Solution” task is called.

Each element chare defines tasks (i) to compute the volume terms on the left-hand side of Eq. (3.10), (ii) to compute, send and receive the data needed by the HRSC limiter (see §3.3), and (iii) to advance the solution forward in time. The asynchronous, message-driven parallelism is achieved by these “Charm++-level” tasks. In turn, each task is composed of traditional routines, such as computing primitive variables

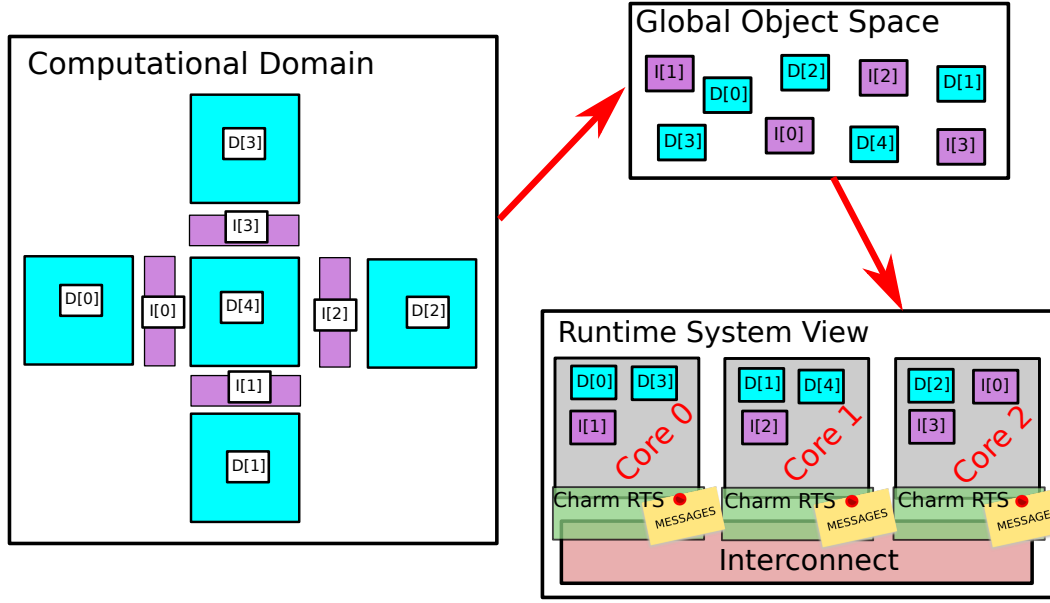


Figure 2: Translation of the DG scheme (3.10) into a parallelized Charm++ application. First, we define the “Computational Domain” as a collection of elements and interfaces with grid data shown in Fig. 1. Within the code, each interface (purple) and similarly each element (cyan) is represented as an element of an array of chares (a chare is a special Charm++ object). The collection of all chares defines a “Global Object Space.” The code defines tasks that need to be performed by these objects as well as the communication pattern between objects. Finally, at runtime, Charm++ creates interface and element chares on available cores to which they are bound. The default distribution is a round-robin mapping, which is shown in the figure. Tasks that can be performed on core 0, for example, are those that are assigned to elements 0 and 3 as well as interface 1. Task creation, routing, and scheduling are handled by the Charm++ runtime system (RTS).

or equation of state evaluations, which are executed sequentially as part of a task’s overall workload. The task dependencies are specified implicitly by the entry method calls appearing throughout the code. Fig. 3 represents these dependencies when taking a single timestep $t \rightarrow t + \Delta t$ on a single element. This graph represents the minimum number of tasks for a basic DG scheme, and more complicated scenarios involving data observations (e.g. reductions) are specified as additional tasks, for example.

4.4. Portability and targeted builds

The Charm++ library can be installed on a variety of architectures and interconnect networks. Since the Charm++ framework naturally decouples low-level parallelization details from SpECTRE’s code, it is straightforward to run SpECTRE in different computing environments while using machine-specific optimizations provided by the Charm++ library. Running on Blue Waters, for example, our code passes messages with direct calls to Cray’s low-level Generic Network Interface (GNI) API for efficient utilization of the Gemini network [79]. We have successfully ran SpECTRE on a variety of other machines supporting a variety of communication libraries/infrastructures including TCP/IP, IB-Verbs (for InfiniBand), and MPI.

We have found the code’s performance to be strongly influenced by how Charm++ is compiled. In order to interpret performance metrics most clearly (cf. §5), we briefly describe those choices that have the greatest influence on performance.

First, one must select a *target architecture* to build Charm++ for. In pure shared-memory environments, such as a desktop, we always use a “multicore” target build. On clusters connected with an InfiniBand network Charm++ can directly use InfiniBand’s IB-Verbs API, which is enabled by compiling as a “verbs-” target build. This is our default configuration on systems that support it. As previously mentioned, Blue Waters uses a

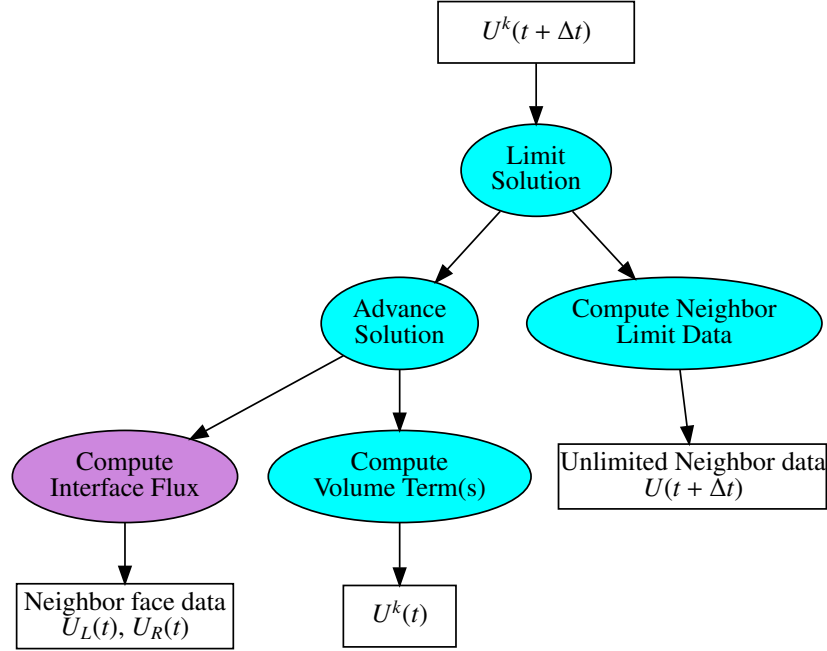


Figure 3: Directed acyclic graph showing high-level task (depicted as ellipses) and data (depicted as rectangles) dependencies for the DG algorithm advancing the numerical solution forward in time on a single element. Tasks are color coded according to whether they belong to interface (purple) or element (cyan) chares. Some tasks are created more frequently as the number of spatial dimensions increases. In d spatial dimensions, $2d$ “Compute Interface Flux” tasks are created during each timestep, for example. The “Unlimited Neighbor data” is computed from the neighbor’s “Advance solution” data.

Gemini network to connect nodes. For performance benefits [80] Charm++ can directly use Cray’s GNI API, which is enabled by compiling as a “gemini-gni-crayxe” target build.

The second choice determines whether or not the Charm++ RTS should use multiple threads. In keeping with Charm++ terminology, we shall refer to the no-threading option as a *non-SMP build* and the multi-threaded option as an *SMP build*.

A non-SMP build is similar to a pure MPI program. On each core we launch a single instance of SpECTRE. Each SpECTRE process is responsible for executing tasks, managing the task-pool, and communicating with all other processes distributed throughout the supercomputer. By comparison, for an SMP build, fewer SpECTRE processes will be launched than available cores. Each SpECTRE process will spawn exactly one thread dedicated for the RTS and communication, and N_{work} threads to perform tasks.

Consider, for example, an SMP build and running on 1 node with 12 cores. We may launch a single process, then spawn a single communication thread on core 0 and 11 worker threads on the remaining cores numbered 1 – 11. Another possibility would be to launch two processes, each of which spawn a single communication and 5 worker threads (giving 12 threads running on all 12 cores). Typically, if a node has N cores and we request N_{com} SpECTRE processes to be launched, there will be N_{com} communication threads (always using 1 per process) bound to N_{com} cores and $N_{\text{work}} = N - N_{\text{com}}$ cores left over to complete tasks.

With fewer cores available to do computational work, at first sight an SMP build does not seem competitive. However, as described above, communication cost is non-negligible. One cannot hope to achieve scalability by assuming that communication cost is near zero, and that one should “ideally” allocate no resources (no cores) to inter-process communication.

The advantages of an SMP build include a reduced memory footprint, since certain data do not need to be duplicated (e.g. global AMR tree metadata), improved core-to-core connectivity since data can be passed via

pointers instead of making copies, and thus increased scheduling efficiency since idle cores can pick up work from their neighbors with less overhead. In practice, we find the SMP build to be essential when scaling to large core counts.

5. Performance tests

As described in §4.4, Charm++ can be built in several different ways, and, in our experience, discovering the “correct” way is often the primary challenge for obtaining good performance on a new machine. §4.4 summarizes those build strategies we have found to work best. These settings are used throughout this section. For all experiments, the Charm++ RTS assigns interface and element chares to the processors using a round-robin mapping as depicted in Fig. 2.

5.1. Efficient core usage and asynchronous task execution

Task-based parallelism relies on asynchronous, non-blocking communication so that cores can remain active. With Projections, a Charm++ analysis tool [81], we investigated these features on a single, shared-memory node of an in-house cluster. We evolved the relativistic MHD system using the cylindrical blast wave solution described in §6.2.3. The computational grid consisted of $50 \times 50 \times 1$ elements each with second-order basis functions and coupled with an LLF numerical flux. The solution was advanced in time using the AB3 stepper. After each step we applied the MUSCL slope limiter and performed a global reduction to compute a pointwise maximum.

We generate time profile graphs for a single core (Fig. 4a) and a single node (Fig. 4b) run. Only the most computationally intensive tasks are shown. We see that the “Compute Interface Flux” (red), “Compute Volume Terms” (blue), and “Limit Solution” (yellow) tasks dominate the simulation’s cost. When these tasks are distributed among the 12 cores, the asynchronous communication and execution model manifests itself as a nonuniform local simulation time. This is clearly seen from the tasks’ gradual dispersion as the simulation proceeds. Idle cores (shown as white) only appear at the very end of the simulation after many of the chares have finished all of their tasks. In this particular example, the single core job took about 74 seconds to complete and the single node job took about 6.6 seconds, demonstrating excellent utilization of all 12 cores and minimal overhead of the Charm++ RTS.

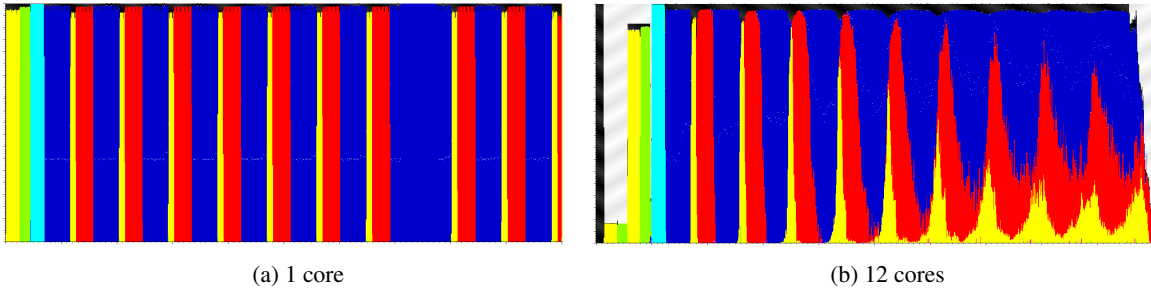


Figure 4: Total amount of time spent executing each task summed across all processors in a given time interval. The vertical axis shows the combined processor utilization (from 0% to 100%) and the horizontal axis shows the wall time. The overhead of the Charm++ RTS is depicted as black and idle core(s) are depicted as white. The additional colors show SpECTRE tasks. **Left:** A typical time profile graph when running on 1 core, where the simulation is perfectly synchronized. **Right:** A typical time profile graph when running on multiple cores, in this case 12. Two important features are immediately apparent. First, the asynchronous nature of task-based parallelism is evident. As the simulation proceeds different parts of the computational grid advance at different rates. Second, all 12 cores remain busy executing their tasks with essentially zero idle time. This would be difficult to achieve within a traditional parallelism framework employing barriers and waits.

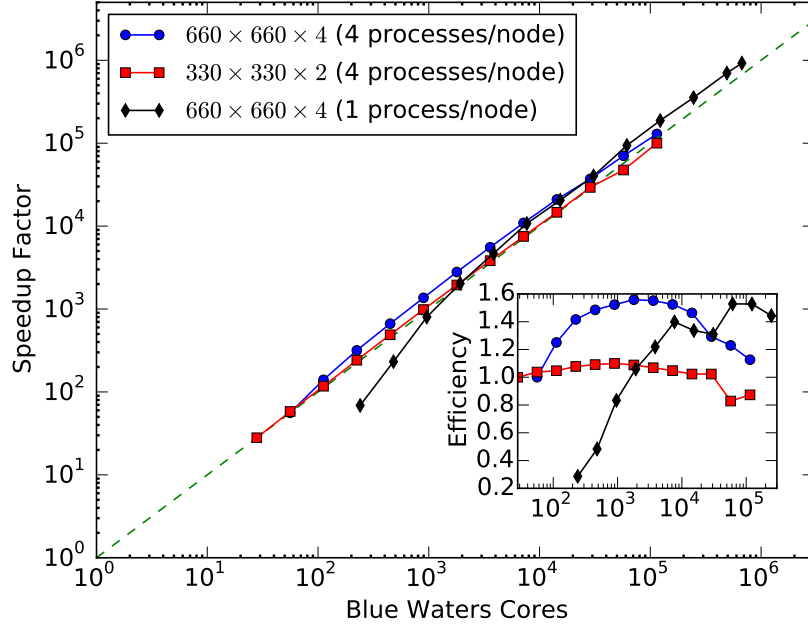


Figure 5: Strong scaling of a SpECTRE evolution of the relativistic MHD system on Blue Waters for different grid sizes and number of SpECTRE processes per node. In all cases, each SpECTRE process spawns a single Charm++ communication thread responsible for passing messages between chares. For the first two cases using 4 SpECTRE processes per node (blue circle and red square), speedups and efficiencies are computed relative to a hypothetical one core job whose walltime is defined to be $T_1 = 28T_{28}$ and where T_{28} is measured directly from a single-node simulation running 28 worker threads. For the cases shown, we measure the walltimes to be $T_{28} \approx 10.101648$ hours (for the $660 \times 660 \times 4$ grid which took 200 simulation steps) and $T_{28} \approx 3.433565$ hours (for the $330 \times 330 \times 2$ grid which took 826 simulation steps). Because simulations using only 1 SpECTRE process per node (black diamond) can be especially inefficient at low core counts (see text), speedups and efficiencies are computed relative to the wall time estimated for a hypothetical single core job obtained by a fitting formula (see text) and found to be $T_1 \approx 712.344370$ hours (which took 200 simulation steps). Similar scalability results (not shown) have been obtained on different machines (e.g. Stampede, Comet, and SciNet) and for different evolution systems.

5.2. Strong scaling

Scaling to a massive number of cores requires sufficient overlap of communication and computation so that work is being done while messages are in flight. This obviously requires non-blocking communication, too. We demonstrate the effectiveness of our code in implementing these requirements by considering a strong scaling test of the relativistic MHD system on Blue Waters using the test problem described in §6.2.4.

To assess scalability, we report parallelization efficiency, E_N , and speedup, S_N , given by

$$E_N = \frac{T_1}{NT_N}, \quad S_N = \frac{T_1}{T_N}, \quad (5.1)$$

where N is the number of “Blue Waters cores” (BW-cores), and T_1 and T_N denote the walltime respectively using 1 and N BW-cores. Every two BW-cores, sometimes also called integer scheduling units or logical cores, share a single floating point unit (FPU). Under ideal circumstances, perfect scalability is achieved whenever $E_N = 1$. In practice, we cannot directly measure T_1 since simulations which run in tens of seconds at large core counts can take weeks or months of walltime on a single core – if the simulation can even fit into available memory. So we instead estimate T_1 . Second, as described below, we sometimes observe values of E_N greater than one suggesting performance bottlenecks for large problems running on small core counts.

Our main result, demonstrating efficient usage of Blue Waters to large core counts, is summarized in Fig. 5. In the figure, the number of BW-cores is equal to the number of worker threads, which gives a modest performance benefit over launching threads on just the floating point units. These results have been obtained with an SMP build of Charm++ (see §4.4). Because of the simulation’s memory requirements, some configurations could not be run at low node counts.

The blue circles and red squares shown in Fig. 5 represent a similar set of configurations. In both cases we have launched 4 SpECTRE processes per node (2 processes per socket) each of which spawn 1 communication thread, leaving 28 BW-cores available on each node to complete tasks. The total number of BW-cores quoted for these cases is 28 times the number of nodes. We ran a sequence of jobs from 1 node (28 BW-cores) to 4,096 nodes (114,688 BW-cores). With a computational grid composed of $330 \times 330 \times 2$ elements, good scalability is observed up to the highest core count where, on average, each core is responsible for only ≈ 2.8 elements. Repeating the experiment after refining the grid to $660 \times 660 \times 4$ elements shows similar scalability, albeit with reduced performance at low core counts (see discussion below). In both cases, we compute measures of scaling performance 5.1 relative to a hypothetical one core job whose walltime is defined to be $T_1 = 28T_{28}$ and where T_{28} is measured directly from a single-node simulation. Note that our definition for T_1 estimates its true value by assuming perfect intra-node scaling.

Beyond 4,096 nodes our simulations exhausted the system’s available memory pages. A possible explanation could be that with more messages in flight, more pages are necessary to accommodate message buffers. We work around this problem by launching fewer processes per node. For these runs, which used just one SpECTRE process (hence just one communication thread) per node, we also left one core free for system processes, leaving 30 threads per node for work. The black diamonds shown in Fig. 5 represent data from this configuration, which continues to show excellent scaling up to the machine’s *entire* pool of 22,380 available nodes (671,400 cores) despite an average of just ≈ 2.8 elements per core. Because simulations using only 1 SpECTRE process per node can be especially inefficient at low core counts (see below), speedups and efficiencies are computed relative to the wall time estimated for a hypothetical single core job obtained by fitting a portion (from 3,840 to 491,520 BW-cores) of the walltime data to a straight line and evaluating this fit at one core.

In all cases considered, we find that whenever the number of chares per process becomes too large, our code becomes inefficient. This could be because we overwhelm the Charm++ RTS with too many messages; the Charm++ documentation and mailing lists mention this as a potential problem. As a future performance improvement we may consider modifying the code to pack more elements into a chare, thereby reducing the number of chares per process.

As a final measure of the code’s performance, we also record the CPU time per timestep per total number of simulation grid points (e.g. the first sequence of simulations uses $660 \times 660 \times 4 \times 2^3$ points, about 14 million). Here the CPU time is computed as the wall time (which is directly measured) multiplied by the number of floating point units being utilized by the worker threads (two worker threads share a single floating point unit). At large core counts, jobs using 4 processes per node behave similarly, with e.g. $\approx 110\mu\text{s}$ needed per step per grid point when running on 896 cores. Jobs using 1 process per node require $\approx 300\mu\text{s}$ when running on 61,440 cores.

5.3. Weak scaling

Figure 6 demonstrates the code’s scalability on the San Diego Supercomputer Center’s machine Comet. Each compute node features two Intel Xeon E5-2680v3 2.5 GHz chips, each equipped with 12 cores, for a total of 24 cores per node. Nodes are connected by an InfiniBand interconnect.

Having just shown strong scalability, we now report on a weak scaling experiment on Comet whereby the number of elements per core is held fixed as the number of nodes is increased from 1 (24 cores) to 72 (1,728 cores), the maximum allowable amount on this machine. Computational grids with heavy (8 elements per core with degree $N = 3$ basis functions) and light (1 element per core with degree $N = 1$ basis functions) workloads show similar trends up to this machine’s largest allowable core allocation. Non-SMP jobs use 24 processes per node and SMP jobs launch 4 communication threads per node leaving 20 worker cores available.

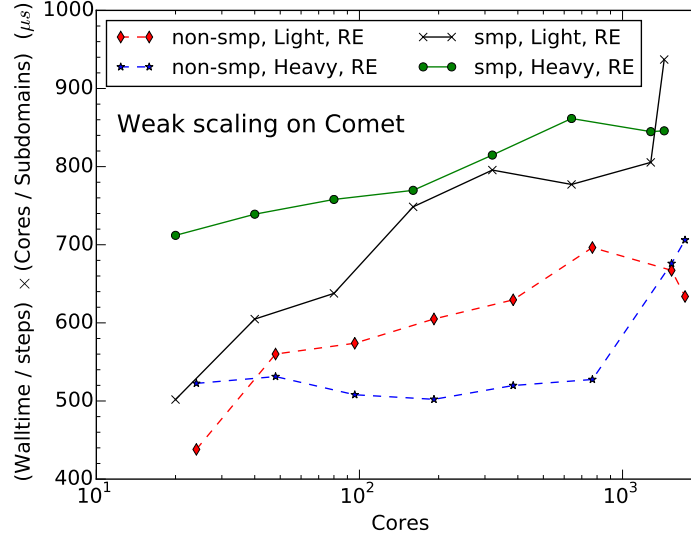


Figure 6: Weak scaling of a SpECTRE evolution of the relativistic hydrodynamics system on Comet with (SMP) and without (non-SMP) multithreading. The problem setup is identical to the smooth flow convergence test described in §6.2.2. We consider computational grids with a “heavy” and “light” amount of work, and report a CPU time (lower is better) which has been scaled by the number of elements to facilitate the comparison of different grid resolutions. Similar scalability results (not shown) have been obtained on different machines (e.g. Stampede and SciNet) and for different evolution systems.

6. Benchmark tests

In this section we present a number of useful benchmark tests in one and three spatial dimensions. All of our tests report on simulations of the Newtonian Euler (NE), Relativistic Euler (RE), and Relativistic MHD (RMHD) systems. Our goal is both to demonstrate that the numerical scheme works as expected and to perform comparisons with other codes or exact solutions. Except for a few special instances, we shall largely avoid comparing the accuracy and efficiency of numerical schemes that differ by choice of, for example, the numerical flux. We shall sometimes use the shorthand tag {Numerical Flux}-{Slope Limiter}-{GLL points per dimension per element} to refer to a particular numerical scheme. For example, “Roe-MRS50-3” would denote a simulation using a Roe numerical flux with 3 GLL points along each dimension (basis functions of degree 2 polynomials) and an MRS limiter with $\tilde{\alpha} = 50$.

Unless stated otherwise, we report the numerical error as an L_1 norm,

$$\|u - v\|_{L_1} = \sum_{i=0}^{n-1} \int_{\Omega} |u_i - v_i| dV. \quad (6.1)$$

Here u and v are the analytic and numerical solution vectors. The integral is computed using a Gauss-Legendre-Lobatto quadrature rule (see Algorithm 25 from [55]).

6.1. One-dimensional tests

6.1.1. Smooth flow (RE) and comparison to Bugner et al.

Our first test compares to the code of Bugner et al [32] by replicating their “smooth sine wave” special relativistic hydrodynamics test (problem 51 in Bugner). For this one-dimensional test, their code is algorithmically equivalent to ours if their mass matrix is computed with the same Gauss-Lobatto technique described in §3.1 so that it is diagonal.

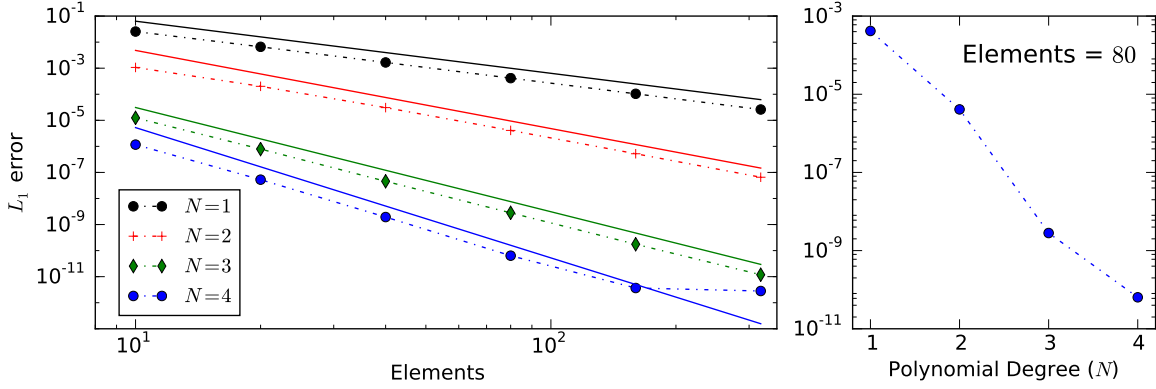


Figure 7: Convergence of the numerical solution for a sequence of simulations performed with the relativistic hydrodynamics system and described further in §6.1.1. We consider convergence by increasing the number of elements (left) and polynomial degree of the numerical approximation (right). These error profiles are typical of our DG scheme when the solution is smooth. In this case the DG scheme achieves exponential convergence in the approximation error as the polynomial degree N is increased. For a fixed value of N , the approximation error decreases with a power law (dashed line) at a rate which closely matches the expected rate of $-(N+1)$ (solid line). The data plotted here is also given in Table B.1.

Name	System	ρ_L	u_L	P_L	ρ_R	u_R	P_R	Γ	x_d
Sod	Euler	1.0	0.0	1.0	0.125	0.0	0.1	1.4	0.0
Shock tube 1	SR Euler	10.0	0.0	13.33	1.0	0.0	10^{-8}	5/3	0.5
Shock tube 2	SR Euler	1.0	0.0	1000.0	1.0	0.0	10^{-2}	5/3	0.5

Table 1: Initial data for Riemann problem tests. In all cases, the fluid is taken to be an ideal gas with an adiabatic index Γ and the initial discontinuity located at x_d .

To repeat their test, we consider the 1-dimensional analog of Eq. (6.4) using a setup identical to theirs and described in the caption of Table B.1. Our errors are quoted in Table B.1. Comparing to their L_1 -errors [82] we find excellent agreement with their code, with relative differences smaller than 5×10^{-12} . These results constitute a strong, independent sanity check for both codes.

For the smooth solution considered here, Fig. 7 shows the expected spectral (exponential) convergence in the approximation. That is, for a fixed number of uniform elements, as we increase the polynomial degree N , the error (6.1) typically decreases like $\sim \exp(-cN)$ for some positive constant c . This property of a DG scheme is observed in all tests whose solution is smooth enough.

6.1.2. Riemann problems (NE)

Riemann problems comprise an important class of tests for a hydrodynamics code. These one-dimensional problems specify constant initial data for the left, U_L , and right, U_R , states

$$U = \begin{cases} U_L, & \text{for } x \leq x_d \\ U_R, & \text{for } x > x_d \end{cases} \quad (6.2)$$

on either side of a discontinuous interface, which we have taken to be located at $x = x_d$. Depending on the values for the left and right states, this initial data may evolve into some combination of a contact discontinuity, shock, and rarefaction wave. The specific wave-like configuration is computable from the initial data [83], which in turn permits the computation of an exact solution [84].

For the Newtonian Euler system we shall consider the standard Sod shock tube test whose initial data is given in Table. 1. Our purpose is to verify basic properties of the numerical scheme and its ability to resolve

sharp features. Appendix A explores the strengths and weaknesses of different limiters and numerical fluxes and provides additional details. In all cases, our numerical scheme exhibits a convergence rate of ≈ 1 (see Table A.1), as expected for discontinuous solutions of this type. A visual inspection (see Figure A.1) of the solution reveals, broadly speaking, that the Roe flux outperforms all three of its competitors while the Local Lax Friedrichs flux typically performs the worst.

6.1.3. Riemann problems (RE)

We now consider two Riemann problems (6.2) in the context of special relativistic hydrodynamics, again comparing our numerical solution to an exact solution [85]. These are standard tests frequently used to verify relativistic hydrodynamic codes (e.g. [86, 87]).

The tests we consider here are more challenging than the Sod shock tube problem of §6.1.2. Numerical errors can lead to situations where the density becomes negative or there is no solution to the equations expressing the primitive variables in terms of the conserved ones. To remedy these problems, in low-density regions we follow the prescription described in the “Atmosphere treatment” appendix of Ref. [88], which guarantees a solution exists with physically reasonable bounds on the primitive variables.

The first Riemann problem we consider is “shock tube 1” whose initial data is given in Table 1. The solution develops a left-moving rarefaction wave, a right-moving contact wave, and a right-moving shock wave. The fluid’s velocity reaches a mildly-relativistic maximum of ≈ 0.7 . The expected order of convergence (see Table B.2) and good resolution of the discontinuous features (see Figure 8) are observed over a wide range of numerical fluxes and limiters. A few of the ≈ 50 tests we ran on this problem failed. Among the four possible fluxes, the Marquina flux failed most often, especially when paired with the MRS limiter. The MRS limiter failed more often whenever using $\alpha \geq 50$ and $N \geq 2$.

The next Riemann problem we consider is “shock tube 2” whose initial data is given in Table 1. The solution develops the same wave structure as “shock tube 1”, but now the fluid’s velocity reaches into the relativistic regime. The shock’s speed, for example, is ≈ 0.986 corresponding to a Lorentz factor of about 6 [86]. An extremely thin “blast wave” of high density develops between the contact discontinuity and the shock. Because of the blast wave’s narrow width, ≈ 0.01 at $t = 0.4$ [87], most codes have difficulty resolving this solution except at very high grid resolutions [86, 87]. At high resolutions our DG scheme is able to capture the features of this solution (Figure 8) and we observe convergence a bit lower than first order (see Table B.2), which has also been observed in finite volume codes running this test [87]. All ≈ 50 numerical schemes we tried were stable for this test case.

6.1.4. Riemann problems (RMHD)

To test the ability of our code to handle shocks when using the relativistic equations of magnetohydrodynamics (RMHD), we first consider a standard suite of 1D RMHD shock tests due to Komissarov [89]. These tests are all performed in Minkowski spacetime. The initial configuration consists of a left state U^L for $x < 0$ and a right state U^R for $x > 0$, each with constant fluid variables. The initial conditions for the different tests are listed in Table 2. All tests use an ideal gas equation of state $P = (\gamma - 1)\epsilon\rho$, with $\gamma = 4/3$. We note that although the numerical domain in these tests is one-dimensional, the evolved variables (velocity, magnetic field) are three-dimensional. For all tests, we use a resolution $\Delta x = 0.01$, and a third-order Runge-Kutta time stepper.

Results for the SlowShock test and the ShockTube1 test are shown in Fig. 9. The results of these tests are similar to those some of us obtained with a finite volume, relativistic MHD code in [88]. The only exceptions are when using the MRS limiter with more than 2 GLL points per element, and for the FastShock test. For high-order elements, the MRS limiter with $\alpha > 100$ does not appear to be dissipative enough to stably evolve shocks. The FastShock, on the other hand, evolves towards values of the conservative variables that do not correspond to any physical set of primitive variables. After correction of those variables following the methods described in [88], the shock front propagates at a speed 2% – 5% slower than the expected value, unless very aggressive limiting methods are used (e.g., setting all points within a DG element to the same value as soon as limiting is required). The exact propagation speed depends on the choice of limiter, but converges to the wrong value for our standard limiters. This indicates that new techniques may have to be developed for the evolution of strongly magnetized, relativistic flows. We note, however, that the FastShock test has a Lorentz factor $\Gamma \sim 25$ and a plasma parameter $\beta = 2P/b^2 \sim 0.002$. This is a regime in which global simulations of binary mergers

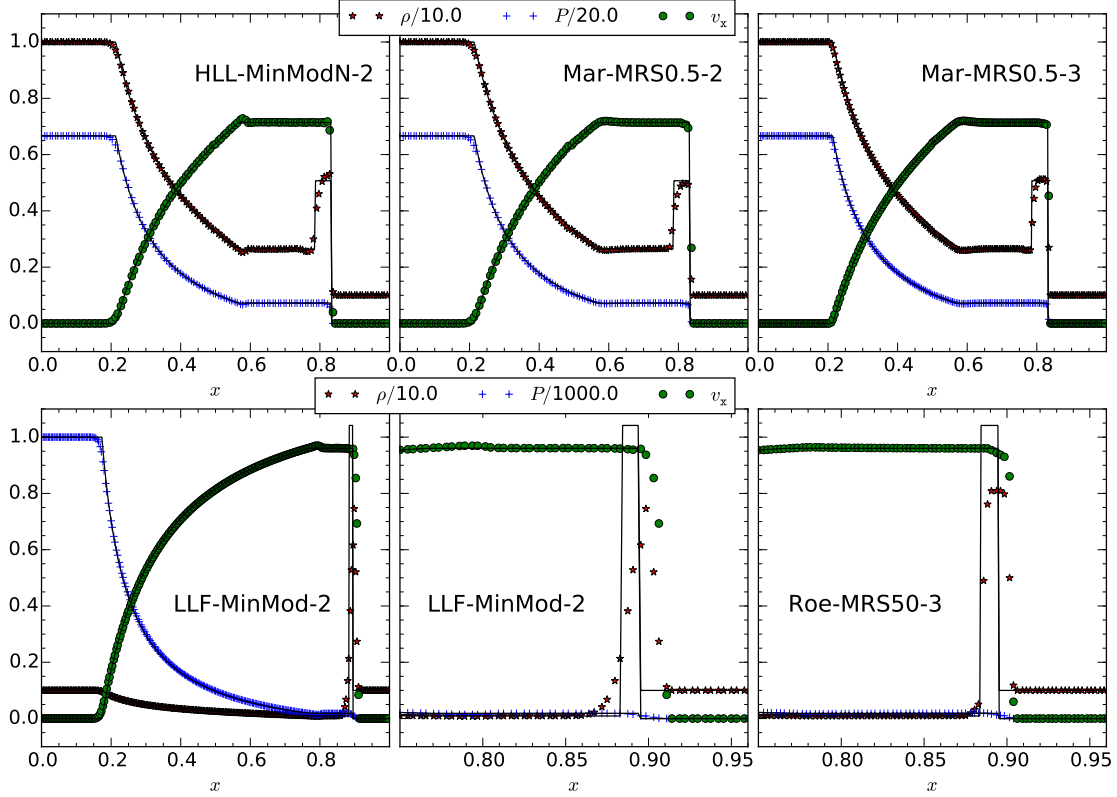


Figure 8: Comparison of the special relativistic shock tube problems computed with different limiters and numerical fluxes. The setup is described in the caption of Table. B.2 and in both cases we evolve on $n_x = 640$ uniform DG elements to $T = 0.4$. **Top row**: Solution to shock tube 1 problem. **Bottom row**: Solution to the highly relativistic shock tube 2 problem. The middle and right panels zoom-in on the sharp blast wave feature that makes this a particularly difficult problem.

Table 2: Initial data for the RMHD shock tests based on Komissarov [89].

Test	Initial state for $x < 0$	Initial state for $x > 0$
Fast shock ($t_{\text{final}} = 2.5$)	$\rho = 1, P = 1$ $u_i = (25, 0, 0), B^i = (20, 25.02, 0)$	$\rho = 25.48, P = 367.5$ $u_i = (1.091, 0.3923, 0), B^i = (20, 49, 0)$
Slow shock ($t_{\text{final}} = 2.0$)	$\rho = 1, P = 10$ $u_i = (1.53, 0, 0), B^i = (10, 18.28, 0)$	$\rho = 3.323, P = 55.36$ $u_i = (0.9571, -0.6822, 0), B^i = (10, 14.49, 0)$
Switch-off ($t_{\text{final}} = 1.0$)	$\rho = 0.1, P = 1$ $u_i = (-2, 0, 0), B^i = (2, 0, 0)$	$\rho = 0.562, P = 10$ $u_i = (-0.212, -0.590, 0), B^i = (2, 4.71, 0)$
Switch-on ($t_{\text{final}} = 2.0$)	$\rho = 0.00178, P = 0.1$ $u_i = (-0.765, -1.386, 0), B^i = (1, 1.022, 0)$	$\rho = 0.01, P = 1$ $u_i = (0, 0, 0), B^i = (1, 0, 0)$
Shock tube 1 ($t_{\text{final}} = 1.0$)	$\rho = 1, P = 1000$ $u_i = (0, 0, 0), B^i = (1, 0, 0)$	$\rho = 0.1, P = 1$ $u_i = (0, 0, 0), B^i = (1, 0, 0)$
Shock tube 2 ($t_{\text{final}} = 1.0$)	$\rho = 1, P = 30$ $u_i = (0, 0, 0), B^i = (0, 20, 0)$	$\rho = 0.1, P = 1$ $u_i = (0, 0, 0), B^i = (0, 0, 0)$
Collision ($t_{\text{final}} = 1.22$)	$\rho = 1, P = 1$ $u_i = (5, 0, 0), B^i = (10, 10, 0)$	$\rho = 1, P = 1$ $u_i = (-5, 0, 0), B^i = (10, -10, 0)$

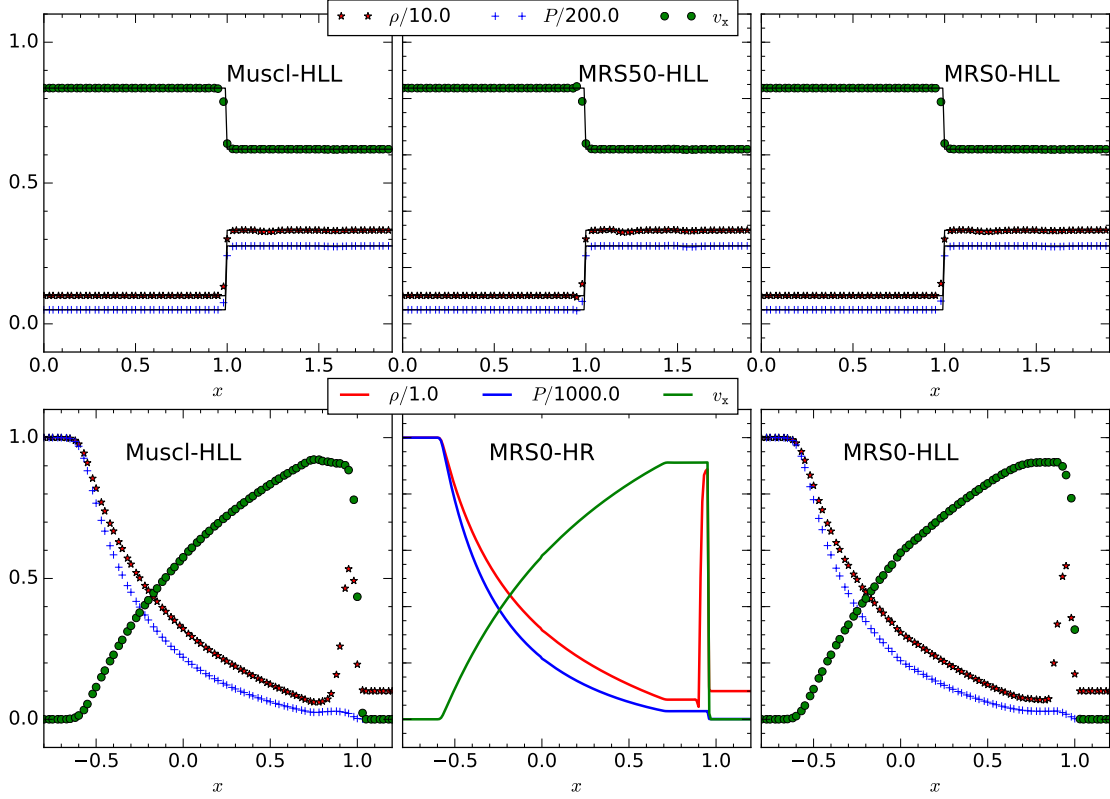


Figure 9: Comparison of the special relativistic MHD shock tube problems computed with different limiters. We show the SlowShock (top) and ShockTube1 (bottom) tests described in Table 2. We solve the test on uniform elements with grid spacing $\Delta x = 0.01$ and 2 GLL points per DG element. For the SlowShock test, the analytical solution is shown as a thin black line. For the ShockTube1 test, we show instead a high resolution simulation ($\Delta x = 0.001$) in the middle panel.

and accretion disks no longer try to accurately evolve the GRMHD system of equations, and instead impose an upper bound on Γ and/or a lower bound on the density (artificial atmosphere) [see e.g. [88]].

6.2. Three-dimensional tests

6.2.1. Isentropic vortex (NE)

The isentropic vortex solution ² is given by

$$\rho = \left(1 - \frac{(\gamma - 1)\beta^2}{8\gamma\pi^2} e^{1-r^2}\right)^{1/(\gamma-1)}, \quad v_x = U - \tilde{y} \frac{\beta}{2\pi} e^{(1-r^2)/2}, \quad v_y = V + \tilde{x} \frac{\beta}{2\pi} e^{(1-r^2)/2}, \quad v_z = W, \quad (6.3)$$

where $\tilde{x} = x - X_0 - Ut$, $\tilde{y} = y - Y_0 - Vt$, and $r^2 = \tilde{x}^2 + \tilde{y}^2$. The constants U , V , and W specify the velocity of the mean fluid flow, X_0 and Y_0 locate the vortex center at $t = 0$, β sets the vortex strength, and γ is the polytropic index for the adiabatic equation of state $P = \rho^\gamma$. Our numerical experiment, fully described in the caption of table B.3, varies both the polynomial degree and number of elements. The resulting numerical errors, quoted in Table B.3, exhibit the expected convergence rate for a fixed polynomial degree N and spectral convergence with N .

²The vortex solution presented in Ref. [90] and its generalization in Ref. [17] contain typos. A corrected expression can be found in the online errata for Ref. [17].

N_C	$K = 1$	$K = 2$	$K = 4$	$K = 8$	$K = 16$	$K = 32$	$K = 64$
2	3.5e-01	3.5e-01 (0.0)	3.1e-01 (0.2)	1.6e-01 (1.0)	4.4e-02 (1.8)	1.1e-02 (2.0)	2.8e-03 (2.0)
3	3.5e-01	1.6e-01 (1.2)	1.6e-02 (3.2)	3.8e-03 (2.1)	7.6e-04 (2.3)	1.2e-04 (2.6)	1.7e-05 (2.9)
4	3.0e-01	5.7e-02 (2.4)	3.5e-03 (4.0)	1.0e-04 (5.1)	5.3e-06 (4.3)	3.4e-07 (3.9)	2.1e-08 (4.1)
5	1.8e-01	4.3e-03 (5.4)	1.4e-04 (5.0)	6.9e-06 (4.3)	3.3e-07 (4.4)	1.3e-08 (4.7)	4.3e-10 (4.9)
6	1.1e-01	1.3e-03 (6.4)	1.2e-05 (6.7)	9.6e-08 (7.0)	1.6e-09 (5.9)	—	—

Table 3: h -convergence data for the relativistic hydrodynamics solution (6.4) with $A = 0.2$, $\gamma = 5/3$, $(U, V, W) = (0.2, 0.2, 0.2)$, and $k = 2\pi$. We report L_1 -errors computed at $T = 2$ with the local convergence order in parentheses. The computational domain is periodic on $(x, y, z) \in [0, 1]^3$. Each convergence test uses N_C GLL collocation points per dimension per element and varies the numerical resolution using $K \times K \times K$ elements with K running from 2 to 64. The numerical solution is advanced forward in time with RK3-SSP and using $\Delta t = 1.7 \times 10^{-4}$, which results in a negligible temporal discretization error. We show results using the Local Lax-Friedrichs numerical flux; other flux choices are qualitatively similar.

6.2.2. Smooth flow (RE)

We consider a relativistic generalization of the periodic, smooth flow solution of Xu et al. [91] given by

$$\rho = 1 + A \sin(u), \quad \epsilon = \frac{1}{(\gamma - 1)\rho}, \quad v_x = U, \quad v_y = V, \quad v_z = W, \quad (6.4)$$

where $u = k[x + y + z - (v_x + v_y + v_z)t]$. The constants U , V , and W specify the velocity of the mean fluid flow, the constant $A \leq 1$ describes the size of the fluid's oscillation about unity, and k is the wave number. This choice of specific internal energy is arranged to give a constant pressure, $P = (\gamma - 1)\rho\epsilon = 1$. Our numerical experiment, fully described in the caption of Table 3, varies both the polynomial degree and number of elements. The resulting numerical errors, quoted in Table 3, exhibit the expected convergence rate for a fixed polynomial degree N and spectral convergence with N .

6.2.3. Cylindrical blast wave (RMHD)

The magnetized cylindrical blast wave test provides a test of a single strong shock propagating in an asymmetric manner in a two-dimensional domain. We evolve the system in 3D, assuming symmetry along the z -axis. We consider an ideal gas equation of state $P = (\gamma - 1)\epsilon\rho$ with $\gamma = 4/3$. At the initial time, we have high-density hot material at rest within a cylinder of radius $r < 0.8$, and low-density cold material at rest at $r > 1$, with a smooth transition between the two regions. The system is initially threaded by a constant magnetic field $\vec{B}^i = (0.1, 0, 0)$. The inner region has $\rho = 0.01$, $P = 1$, while the outer region has $\rho = 0.0001$, $P = 0.0005$. In the transition region $0.8 < r < 1$, we perform linear interpolation in the logarithm of ρ and P . The resulting evolution is a blast wave which, because of the asymmetry introduced by the magnetic field, propagates faster along the x -axis than the y -axis. The computational domain covers $x = [-6, 6]$, $y = [-6, 6]$, and uses $100^2 \times 1$ elements with 2 GLL points along each dimension in each element.

We show results at $t = 4$ in Fig. 10, using the MRS limiter with $\alpha = 100$. These results can be compared with finite volume results using the same number of grid points (200×200) [92]. Our results are generally smoother upstream of the blast wave and in the high density regions, but show features associated with the grid structure behind the shock that are not present in the finite volume simulations. If we perform the same simulation using the MUSCL limiter (see Fig. 11), the unphysical features behind the shock disappear, but the overall quality of the solution is significantly lower. This is because the MUSCL limiter aggressively limits the solution within each DG element, causing the code to effectively behave like a lower-resolution finite-volume scheme in the presence of shocks.

6.2.4. Orszag-Tang vortex (RMHD)

We now consider the evolution of the Orszag-Tang vortex benchmark test [93], which features shock-shock interactions. We use the MRS limiter with $\alpha = 10$. To the best of our knowledge, this test is the first application

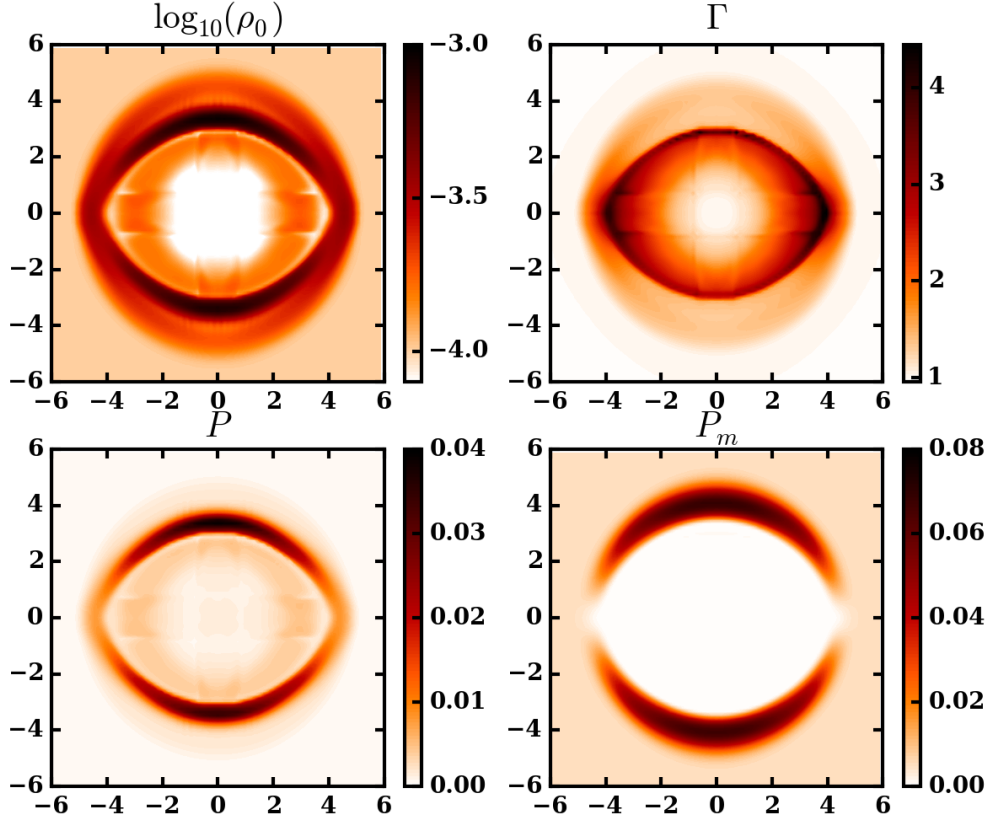


Figure 10: Cylindrical blast wave test at $t = 4$, using the MRS limiter with $\alpha = 100$. The shock wave is accurately evolved, but the grid is visible in some quantities behind the shock.

of the MRS limiter to the (relativistic) MHD system, and so our results further demonstrate the robustness of this limiter for a new, challenging test case.

The Orszag-Tang vortex has been used extensively as a test of MHD codes, e.g. [94–97]. It is set on a 2-dimensional periodic domain with $0 \leq x < 1$ and $0 \leq y < 1$ with a constant initial rest mass density $\rho_0 = 25/(26\pi)$ and specific energy density $\epsilon = 5/(8\pi)$. We use an ideal gas equation of state $P = (\gamma - 1)\epsilon\rho$, with $\gamma = 5/3$. The fluid velocity is $v_x = -0.5 \sin(2\pi y)$, $v_y = 0.5 \sin(2\pi x)$, and the magnetic field is $B_x = -(1/\sqrt{4\pi}) \sin(2\pi y)$, $B_y = (1/\sqrt{4\pi}) \sin(4\pi x)$. Although the Orszag-Tang vortex is a 2D problem, we formally evolve it in 3D (but with only one DG element in the z -direction, with periodic boundary conditions).

In Fig. 12, we show the state of the system after $t = 0.8$, for DG elements of size $\Delta x = \Delta y = \Delta z = 0.01$ and with $2 \times 2 \times 2$ collocation points each. We find results consistent with those of finite volume codes at similar resolution, which is as good as one might expect for a test with multiple strong shocks propagating and interacting.

The Orszag-Tang test is particularly demanding for the divergence cleaning scheme used in SpECTRE. Numerical errors constantly drive the divergence of the magnetic field, particularly at shocks. We see in Fig. 12 that the divergence cleaning scalar at $t = 0.8$ is $\sim 3\%$ of the value of the magnetic field or, maybe more meaningfully, $\sim 0.3\%$ of $|\nabla \mathbf{B}|$. At the same evolution time, the maximum of Φ is $\sim 0.5\%$ of $|\nabla \mathbf{B}|$ for $\Delta x = 0.02$.

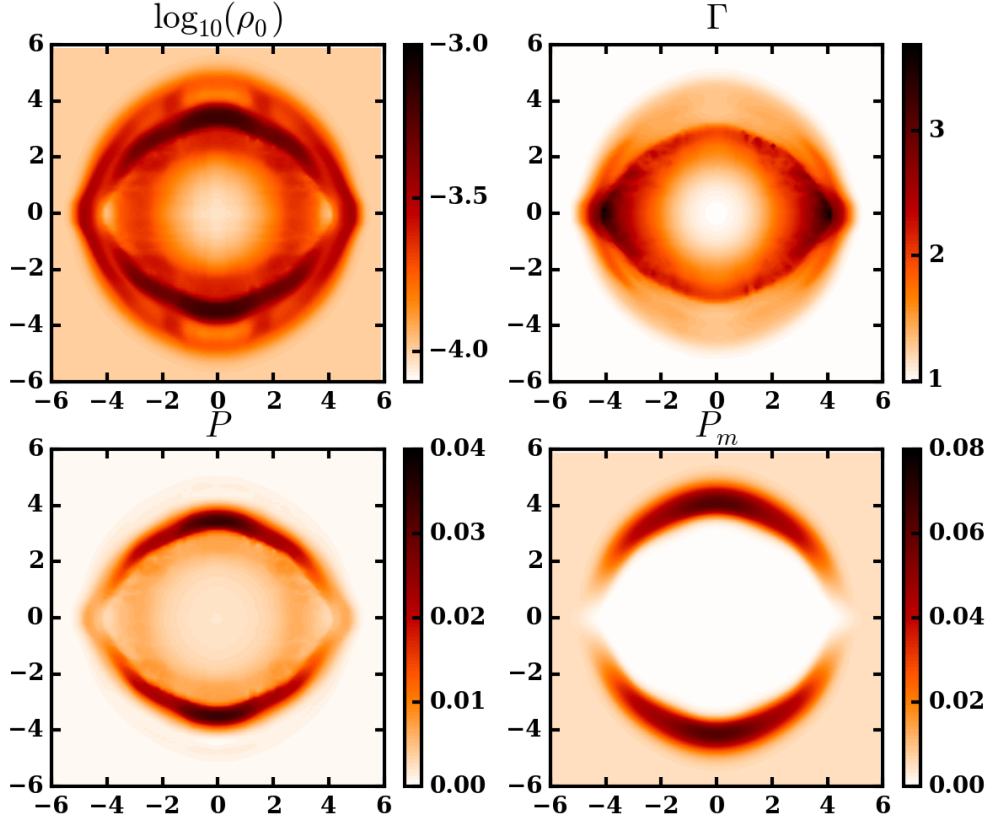


Figure 11: Cylindrical blast wave test at $t = 4$, using the MUSCL limiter. The shock wave is not as sharp as with the high-order MRS limiter, but does not show any structure along the coordinate axis of the grid.

6.2.5. Spherical Bondi accretion at $t = 0$ (GRMHD)

All of the tests considered so far were performed in flat space on a Minkowski metric. To test the general relativistic terms in the GRMHD equations, we consider the well-known steady-state solution for spherical accretion onto a Schwarzschild black hole, due to Michel [98]. Michel's solution is the general relativistic version of spherical Bondi accretion. Conveniently, if one adds a radial magnetic field of the form $\tilde{B}^r = \tilde{B}^0 r^{-2}$ to that solution, it remains a steady-state solution. It is a useful test of the GRMHD equations because many general relativistic source terms are non-zero and have to exactly cancel each other in order to recover a static solution. Here, we consider an accretion flow with its sonic point at $r_s = 8GM/c^2$, an ideal gas equation of state $P = (\gamma - 1)\epsilon\rho$ with $\gamma = 4/3$, a radial magnetic field $B^0 = 0.1$, a black hole mass $M = 1$, and an accretion rate $dM/dt = 1$. As we have not yet implemented excision of a black hole's interior, and have not attempted to evolve singularities within our computational domain, we do not evolve the problem. Instead, we only verify that the time derivative of all variables converges to zero at the initial time.

We consider two different computational domains. First, we use a domain that does not overlap with the sonic point, $x = [4.5, 5.5]r_g$, $y = [-0.5, 0.5]r_g$, and $z = [-0.5, 0.5]r_g$, with $r_g = GM/c^2$. Then we consider a computational domain with $x = [7.5, 8.5]r_g$, which overlaps the sonic point. In both cases, we do not apply any limiting to the solution. We find first-order convergence of the time derivatives when using 2 basis functions per dimension in each DG element, and third-order convergence when using 4 basis functions.

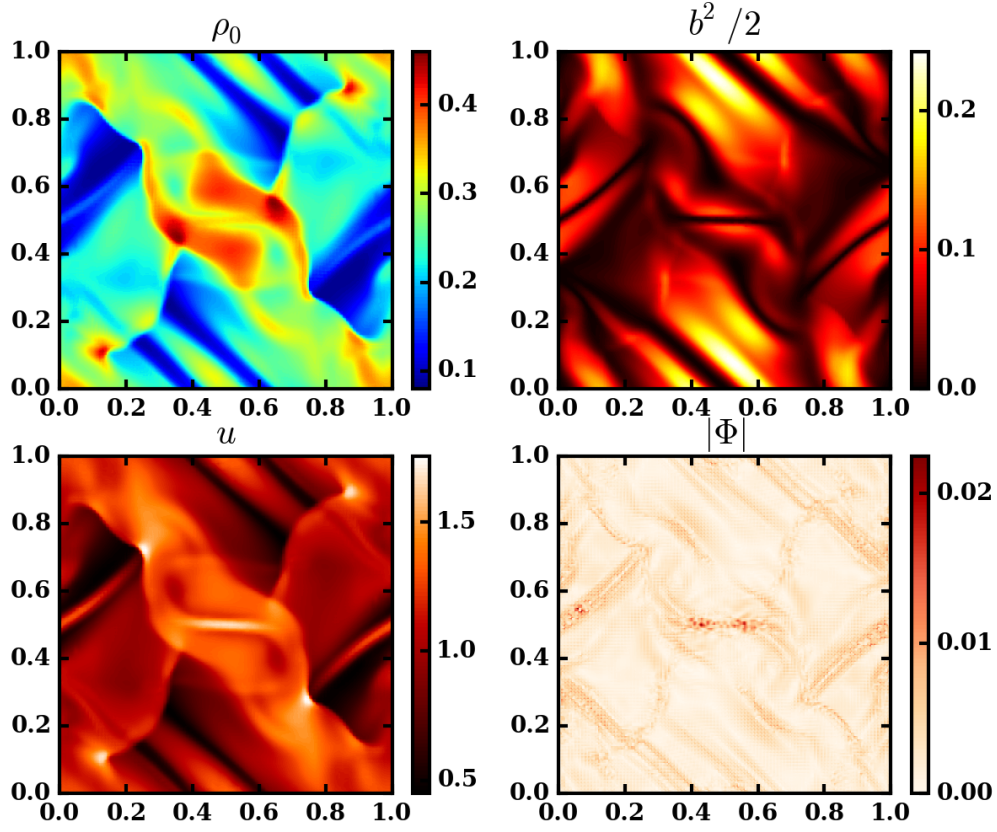


Figure 12: Orszag-Tang vortex at time $t = 0.8$. We show the fluid density (top left), the magnetic pressure (top right), the fluid internal energy (bottom left), and the divergence cleaning scalar Ψ (bottom right).

7. Discussion

We have described a new relativistic astrophysics code SpECTRE. The code differs from other codes in many important respects. In particular, we use a discontinuous Galerkin numerical solver and a task-based parallelism model. To the best of our knowledge, this is the first DG solver parallelized using a task-based model and the first DG treatment of the general relativistic MHD system. As DG methods are comparatively new to astrophysics (see Refs. [30, 32, 33] for recent work), we have provided a detailed description of our numerical scheme including those approximate Riemann solvers and high-resolution shock capturing limiters we have implemented.

Our main results are the performance and benchmark tests given in §5 and §6. In particular, we have shown that the code can solve a wide variety of challenging astrophysics benchmark tests, including highly-relativistic shocks with blast waves (see §6.1.3), shock-shock interactions (see §6.2.4), and the steady-state GRMHD solution of Michel (see §6.2.5). For smooth solutions, we have shown the expected exponential convergence with increasing grid resolution, for example in the case of smooth relativistic flow, §6.2.2. In §5.1 we demonstrate how asynchronous, non-blocking communication promotes efficient resource usage by reducing the amount of idle core time. The scalability experiments of §5.2 demonstrate the code’s performance on large machines. We observe excellent strong scaling on Blue Waters up to the machine’s full capacity of 22,380 nodes using 671,400 worker cores/threads.

Because of the generality and robustness of the DG method, it can serve as the core kernel for the solution

of multi-physics problems that require an ensemble of interacting physical descriptions. Work toward this is underway. To evolve dynamical spacetimes we are implementing the Einstein equations using a generalized harmonic formulation [99, 100] written with first-order time and first-order space derivatives [101]. We consider the fully first-order generalized harmonic system because it has been successfully evolved with pseudospectral methods for a variety of astrophysical configurations [14, 102]. To tackle challenging multi-scale problems, we are adding local time stepping techniques and mesh refinement strategies to either split the elements into smaller elements (h -adaptivity) or increase the number of basis functions in an element (p -adaptivity). The locality of the DG scheme facilitates adaptive resolution refinement and local timestepping, and good results have been shown in other contexts. And while current large codes often struggle to achieve good performance with local time-stepping [103], task-based parallelism and asynchronous communication are promising tools to overcome this problem.

Looking forward, we believe that the benefits of task-based parallelism will become increasingly important in these more complicated contexts. More generally, accurate numerical methods and efficient usage of massively parallel supercomputers will be essential for the high-fidelity simulations needed to realize the promise of current and future experiments.

Acknowledgments

We acknowledge helpful discussions with Sebastiano Bernuzzi, Bernd Brügmann, Marcus Bugner, Mani Chandra, Roland Haas, Daniel Hemberger, Jan Hesthaven, Cameron Hummels, Fraser Hutchison, Jonathan Lifflander, Geoffrey Lovelace, Phil Miller, Harald Pfeiffer, David Radice, and Christian Reisswig. Support for F. Foucart was provided by NASA through Einstein Postdoctoral Fellowship grant numbered PF4-150122 awarded by the Chandra X-ray Center, which is operated by the Smithsonian Astrophysical Observatory for NASA under contract NAS8-03060. Authors at Cornell were partially supported by NSF under award nos. TCAN AST-1333129 and PHY-1306125, and by the Sherman Fairchild Foundation. Authors at Caltech were partially supported by NSF under award nos. TCAN AST-1333520, CAREER PHY-1151197, and PHY-1404569, and by the Sherman Fairchild Foundation. ES acknowledges support from NSF award OCI-0905046. ES and JM were supported by an NSERC Discovery grant. Research at the Perimeter Institute is supported by the Government of Canada through Industry Canada and by the Province of Ontario through the Ministry of Research & Innovation. Computations were performed on NSF/NCSA Blue Waters under allocation PRAC ACI-1440083, on the NSF XSEDE network under allocations TG-PHY100033 and TG-PHY990007, and on the Caltech compute cluster Zwicky (NSF MRI-R² award no. PHY-0960291).

Appendix A. Performance of fluxes and limiters for the Sod problem

In §6.1.2 we considered the standard Sod shock tube test whose initial data is given in Table 1. In this appendix we summarize the approximately 80 h -convergence simulations we performed, from which we draw a few general insights into the relative strengths and weaknesses of different limiters and numerical fluxes. Table A.1 reports the numerical errors compared to the exact solution. In all cases, our numerical scheme exhibits a convergence rate of ≈ 1 , as expected for discontinuous solutions of this type.

Our first numerical test explores the impact of numerical flux choice when using low order elements (degree $N = 1$ basis functions) and a minmod limiter. Numerical errors (see Table A.1) are computed for four different numerical fluxes and show all four schemes perform remarkably similar. The numerical solutions computed with these schemes also appear visually identical; one representative case is shown in the upper-left panel of Fig. A.1. These findings are consistent with those of Qiu and Shu [68].

The next numerical test considers the behavior of the higher-order limiters when using higher-order elements. As demonstrated in Table A.1, these limiters continue to offer accuracy comparable to their lower order counterparts. This is anticipated, since, for a discontinuous solution of this type, linear basis elements are expected to be optimal. Higher elements may provide marginal accuracy benefits at best. At worst, higher elements may lead to large overshoots. Varying the numerical flux, limiter, and polynomial order, we have explored ≈ 80 different convergence tests in total, and find most (reasonable) combinations give consistently similar numerical errors (see Table A.1).

Scheme	$n_x = 20$	$n_x = 40$	$n_x = 80$	$n_x = 160$	$n_x = 320$	$n_x = 640$
HLL-MinMod-2	1.7e-01	9.5e-02 (0.9)	4.9e-02 (1.0)	2.6e-02 (0.9)	1.4e-02 (0.9)	7.7e-03 (0.9)
Roe-MinMod-2	1.7e-01	9.3e-02 (0.9)	4.8e-02 (1.0)	2.6e-02 (0.9)	1.4e-02 (0.9)	7.5e-03 (0.9)
LLF-MinMod-2	1.9e-01	1.1e-01 (0.8)	5.4e-02 (1.0)	2.9e-02 (0.9)	1.5e-02 (0.9)	8.5e-03 (0.9)
Mar-MinMod-2	1.9e-01	1.0e-01 (0.9)	5.3e-02 (1.0)	2.8e-02 (0.9)	1.5e-02 (0.9)	8.0e-03 (0.9)
Roe-MinModN-3	1.2e-01	6.4e-02 (0.9)	3.3e-02 (1.0)	1.7e-02 (1.0)	8.3e-03 (1.0)	4.4e-03 (0.9)
Roe-MRS0.5-3	1.3e-01	6.4e-02 (1.0)	3.3e-02 (1.0)	1.7e-02 (1.0)	8.6e-03 (0.9)	4.6e-03 (0.9)
Roe-MRS5-3	1.1e-01	6.0e-02 (0.9)	3.1e-02 (0.9)	1.6e-02 (1.0)	8.2e-03 (1.0)	4.3e-03 (0.9)
Roe-MRS50-3	6.5e-02	3.9e-02 (0.7)	2.8e-02 (0.5)	1.5e-02 (0.9)	8.0e-03 (0.9)	4.2e-03 (0.9)
Roe-MinModN-4	1.3e-01	6.7e-02 (1.0)	3.5e-02 (0.9)	1.8e-02 (1.0)	8.8e-03 (1.0)	4.6e-03 (0.9)
Roe-MRS0.5-4	1.2e-01	6.7e-02 (0.9)	3.6e-02 (0.9)	1.9e-02 (0.9)	1.1e-02 (0.8)	5.2e-03 (1.0)
Roe-MRS5-4	1.2e-01	6.3e-02 (1.0)	3.2e-02 (1.0)	1.8e-02 (0.9)	9.0e-03 (1.0)	4.9e-03 (0.9)
Roe-MRS50-4	5.7e-02	4.6e-02 (0.3)	2.8e-02 (0.7)	1.5e-02 (0.9)	8.8e-03 (0.8)	4.6e-03 (0.9)

Table A.1: h -convergence data for the NE-Sod problem. We report L_1 -errors computed at $T = 0.25$ with the local convergence order in parentheses. The computational domain $x \in [-0.5, 0.5]$ is divided into n_x uniform elements. The numerical solution is advanced forward in time with RK3-SSP.

Visually inspecting the numerical solution’s “quality” provides another important measure of success. Looking at a $T = 0.25$ snapshot now reveals significant differences between schemes, which are not as easily captured in a numerical error computation. Our most interesting observations are summarized in Fig. A.1 and its caption. When combined with a HRSC limiter, we consistently find that the Roe flux outperforms all three of its competitors while the Local Lax Friedrichs flux typically performs the worst. When used with a Roe flux, we find that the HRSC limiters described in §3.3 perform very well on this test case. Note that the authors of ref. [30] report good shock capturing behavior when using a LLF flux combined with a characteristic variable limiting procedure, which we have not explored here.

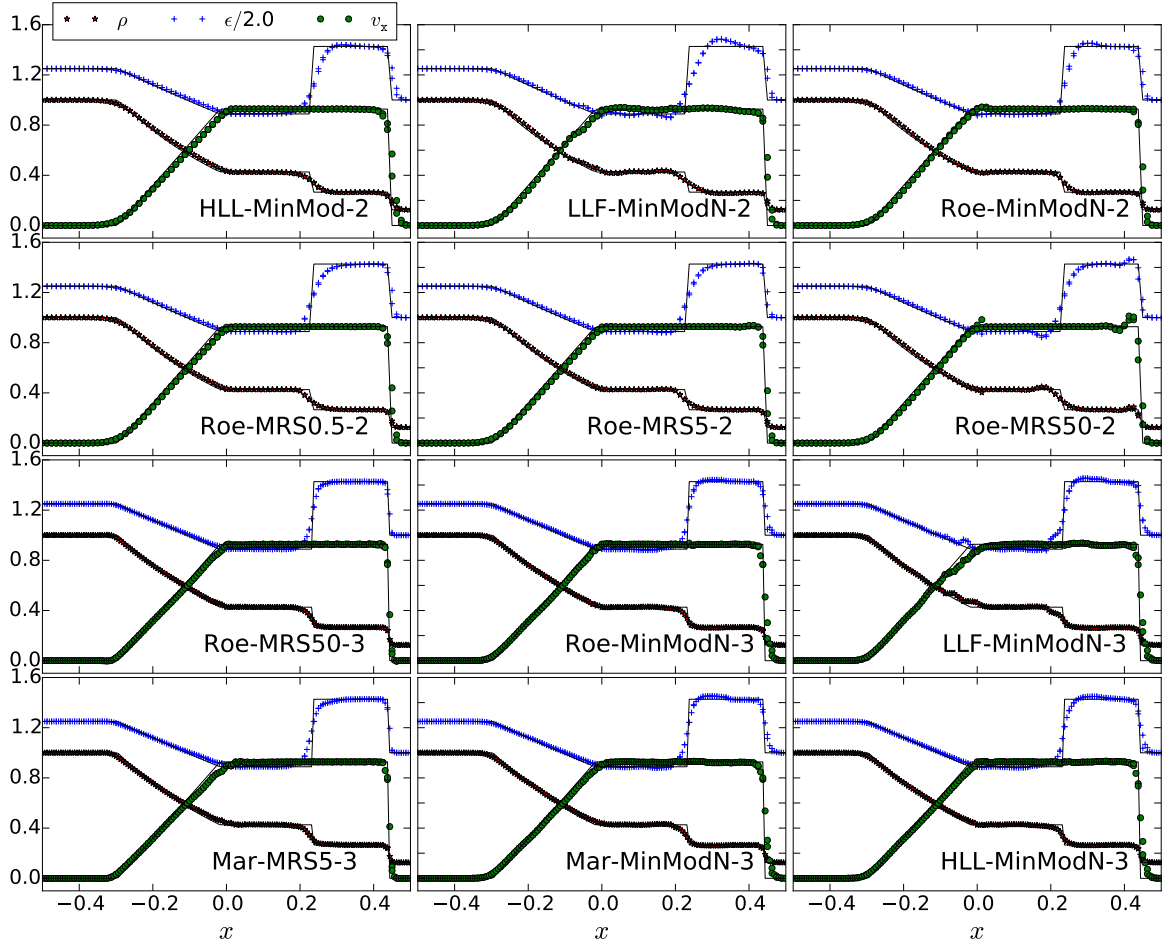


Figure A.1: Comparison of the Sod Shock tube problem computed with different limiters and numerical fluxes. The setup is described in the caption of Table. A.1. We show results with a moderate grid resolution using 80 uniform elements. **Top row:** The first panel shows a typical minmod solution, which is fairly consistent across numerical flux choice. The next two panels show the performance of a high- N minmod limiter. Despite using only first order elements, noticeable oscillations appear in both simulations, with the LLF flux containing significantly more. Qualitatively, the HLL and Marquina fluxes (not shown) appear more similar to the LLF case. **Second row:** These three panels depict a sequence of solutions computed with the parameterized MRS limiter using $\alpha = 0.5, 5, 50$. We observe the limiter to be effective over a large range of values, although as α becomes large the limiter turns off and oscillations return. **Third row:** In the first two panels we see that both the high- N minmod and MRS limiters may perform *better* than their lower-order counterparts. For the LLF flux (third panel) this is not the case, unfortunately. **Bottom row:** The first two panels use a HRSC limiter with the Marquina flux and second order elements. The solution quality is seen to be not quite as good as the comparable simulations using a Roe flux, but still significantly better than the LLF and fairly good overall. The bottom right panel, together with its three neighboring panels, shows all four numerical fluxes using second order elements and the high- N minmod limiter.

Appendix B. Numerical truncation error and convergence tables

A table of numerical data provides an effective means for comparing codes. One particularly important use is to verify that an algorithm has been correctly implemented. Since there are very few relativistic hydrodynamics codes using a discontinuous Galerkin solver, there are correspondingly few tables to compare with. We were also hard pressed to find any tables with which to compare the isentropic vortex (non-relativistic Euler) solution since, in many other fields, the DG scheme is formulated on tetrahedra and with a full mass matrix computed using Gaussian quadratures. As discussed in §3.1, our scheme uses a GLL quadrature rule that results in a diagonal mass matrix [104, 105]. In this appendix we provide a sample of tables of errors and h -convergence data for tests considered in this paper.

N_C	$n_x = 10$	$n_x = 20$	$n_x = 40$	$n_x = 80$	$n_x = 160$	$n_x = 320$
2	2.55e-02	6.60e-03 (1.9)	1.66e-03 (2.0)	4.16e-04 (2.0)	1.04e-04 (2.0)	2.60e-05 (2.0)
3	1.07e-03	2.02e-04 (2.4)	3.07e-05 (2.7)	4.10e-06 (2.9)	5.21e-07 (3.0)	6.54e-08 (3.0)
4	1.24e-05	7.88e-07 (4.0)	4.57e-08 (4.1)	2.81e-09 (4.0)	1.75e-10 (4.0)	1.19e-11 (3.9)
5	1.17e-06	5.27e-08 (4.5)	1.94e-09 (4.8)	6.38e-11 (4.9)	3.67e-12 (4.1)	2.81e-12 (0.4)

Table B.1: h -convergence data for the one-dimensional version of the smooth flow solution (6.4) with $A = 0.2$, $\gamma = 5/3$, $v_x = 0.2$, and $k = 2\pi$. We report L_1 -errors computed at $T = 2$ with the local convergence order in parentheses. The computational domain is periodic on $x \in [0, 1]$ and divided into n_x uniform elements. Each convergence test uses N_C GLL collocation points per dimension per element. The numerical solution is advanced forward in time with RK3-SSP using $\Delta t = 10^{-4}$, which results in a negligible temporal discretization error. We show results using the Local Lax-Friedrichs numerical flux; other flux choices are qualitatively similar. Our results exactly match those produced by the code of Bugner et al (see text) up to $\approx 5 \times 10^{-12}$. We believe that discrepancies observed in smaller values of the numerical error are likely due to a difference in timesteppers and/or the effect of roundoff errors.

Test	Scheme	$n_x = 40$	$n_x = 80$	$n_x = 160$	$n_x = 320$	$n_x = 640$
ST1	HLL-MinMod-2	7.1e-02 (0.7)	4.8e-02 (0.6)	2.6e-02 (0.9)	1.4e-02 (0.9)	7.5e-03 (0.9)
	HLL-MinModN-2	5.8e-02 (0.8)	3.5e-02 (0.7)	1.9e-02 (0.9)	1.0e-02 (0.9)	6.2e-03 (0.7)
	Mar-MRS0.5-2	8.2e-02 (0.6)	5.1e-02 (0.7)	2.5e-02 (1.0)	1.1e-02 (1.2)	5.6e-03 (1.0)
	Roe-MRS5-3	6.8e-02 (0.5)	4.3e-02 (0.6)	1.9e-02 (1.2)	6.5e-03 (1.5)	3.3e-03 (1.0)
		$n_x = 80$	$n_x = 160$	$n_x = 320$	$n_x = 640$	$n_x = 1280$
ST2	LLF-MinMod-2	5.1e-02 (0.7)	3.1e-02 (0.7)	1.9e-02 (0.7)	1.1e-02 (0.7)	6.9e-03 (0.7)
	LLF-MinModN-2	9.8e-02 (0.7)	5.9e-02 (0.7)	3.3e-02 (0.8)	2.1e-02 (0.6)	1.4e-02 (0.6)
	Roe-MRS50-3	2.4e-02 (0.5)	1.5e-02 (0.6)	1.0e-02 (0.6)	7.0e-03 (0.5)	6.2e-03 (0.2)
	Mar-MRS5-2	4.3e-02 (0.7)	2.5e-02 (0.8)	1.6e-02 (0.6)	1.1e-02 (0.6)	7.3e-03 (0.6)

Table B.2: Sample of h -convergence data for the relativistic Euler Riemann problems shock tube 1 (ST1) and shock tube 2 (ST2). Cases not shown are qualitatively similar. The ST2 test is the more challenging of the two, with a narrow blast wave feature and Lorentz factors of about 6. Because of the large size of the exact solutions, we report *relative* L_1 -errors computed at $T = 0.4$ with the local convergence order in parentheses. The normalization factor is simply the exact solution's norm at the final time as computed on the numerical grid, $\|u\|_{L_1} \approx 21$ for ST1 and $\|u\|_{L_1} \approx 1, 151$ for ST2. The computational domain $x \in [0, 1]$ is divided into n_x uniform elements. The numerical solution is advanced forward in time with RK3-SSP. To check our implementation of the various limiters and numerical fluxes, we have performed a total of ≈ 50 convergence studies for each test and find the numerical error for comparable values of n_x to be at most a factor of a few different from the cases shown here.

N_C	$K = 8$	$K = 16$	$K = 32$	$K = 64$	$K = 128$	$K = 256$
2	1.51e+01	5.59e+00 (1.4)	1.73e+00 (1.7)	4.62e-01 (1.9)	1.17e-01 (2.0)	2.93e-02 (2.0)
3	2.92e+00	6.29e-01 (2.2)	1.02e-01 (2.6)	1.78e-02 (2.5)	3.21e-03 (2.5)	5.83e-04 (2.5)
4	1.00e+00	9.73e-02 (3.4)	6.70e-03 (3.9)	3.44e-04 (4.3)	2.15e-05 (4.0)	1.41e-06 (3.9)
5	2.37e-01	7.98e-03 (4.9)	3.13e-04 (4.7)	1.44e-05 (4.4)	6.94e-07 (4.4)	3.66e-08 (4.2)
6	4.95e-02	1.37e-03 (5.2)	2.21e-05 (5.9)	3.03e-07 (6.2)	5.38e-09 (5.8)	2.45e-10 (4.5)
7	1.15e-02	9.88e-05 (6.9)	9.32e-07 (6.7)	1.10e-08 (6.4)	2.68e-10 (5.4)	—
8	2.21e-03	1.41e-05 (7.3)	7.46e-08 (7.6)	3.57e-10 (7.7)	—	—

Table B.3: h -convergence data for the isentropic vortex solution (6.3) with $\beta = 5$, $X_0 = Y_0 = 4$, $(U, V, W) = (1, 1, 0)$, and $\gamma = 1.4$. We report L_1 -errors computed at $T = 2$ with the local convergence order in parentheses. The physical domain is taken to be the slab $(x, y) \in [0, 10]$ and $z \in [0, 1]$ with analytic outer boundary conditions. Each convergence test uses N_C GLL collocation points per dimension per element and varies the numerical resolution using $K \times K \times 1$ elements with K running from 8 to 256. The numerical solution is advanced forward in time with RK3-SSP using $\Delta t = 1 \times 10^{-4}$, which results in a temporal discretization error of $\approx 2 \times 10^{-10}$. We show results using the LLF numerical flux; other flux choices lead to qualitatively similar results.

References

References

- [1] D. Eichler, M. Livio, T. Piran, D. N. Schramm, Nucleosynthesis, neutrino bursts and gamma-rays from coalescing neutron stars, *Nature* 340 (1989) 126. doi:10.1038/340126a0.
- [2] R. Narayan, B. Paczynski, T. Piran, Gamma-ray bursts as the death throes of massive binary stars, *Astrophys. J. Lett.* 395 (1992) L83. arXiv:astro-ph/9204001, doi:10.1086/186493.
- [3] R. Mochkovitch, M. Hernanz, J. Isern, X. Martin, Gamma-ray bursts as collimated jets from neutron star/black hole mergers, *Nature* 361 (1993) 236. doi:10.1038/361236a0.
- [4] H.-T. Janka, Explosion Mechanisms of Core-Collapse Supernovae, *Ann. Rev. Nuc. Part. Sc.* 62 (2012) 407. arXiv:1206.2503, doi:10.1146/annurev-nucl-102711-094901.
- [5] A. Burrows, Colloquium: Perspectives on core-collapse supernova theory, *Rev. Mod. Phys.* 85 (2013) 245. arXiv:1210.4921, doi:10.1103/RevModPhys.85.245.
- [6] C. D. Ott, Massive Computation for Understanding Core-Collapse Supernova Explosions, *Computing in Science & Engineering* 18 (5) (2016) 78–92. arXiv:1608.08069, doi:10.1109/MCSE.2016.81.
- [7] Fermi Gamma-Ray Telescope, <http://fermi.gsfc.nasa.gov>.
- [8] <http://www.ligo.caltech.edu>.
- [9] B. Fryxell, K. Olson, P. Ricker, F. Timmes, M. Zingale, et al., FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes, *Astrophys. J. Suppl. Ser.* 131 (2000) 273. doi:10.1086/317361.
- [10] G. L. Bryan, M. L. Norman, B. W. O’Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman, B. Smith, R. P. Harkness, J. Bordner, J.-h. Kim, M. Kuhlen, H. Xu, N. Goldbaum, C. Hummels, A. G. Kritsuk, E. Tasker, S. Skory, C. M. Simpson, O. Hahn, J. S. Oishi, G. C. So, F. Zhao, R. Cen, Y. Li, Enzo Collaboration, ENZO: An Adaptive Mesh Refinement Code for Astrophysics, *Astrophys. J. Suppl. Ser.* 211 (2014) 19. arXiv:1307.2265, doi:10.1088/0067-0049/211/2/19.

- [11] A. Mignone, C. Zanni, P. Tzeferacos, B. van Straalen, P. Colella, G. Bodo, The PLUTO Code for Adaptive Mesh Computations in Astrophysical Fluid Dynamics, *Astrophys. J. Suppl. Ser.* 198 (2012) 7. arXiv:1110.0740, doi:10.1088/0067-0049/198/1/7.
- [12] A. S. Almgren, V. E. Beckner, J. B. Bell, M. S. Day, L. H. Howell, C. C. Joggerst, M. J. Lijewski, A. Nonaka, M. Singer, M. Zingale, CASTRO: A New Compressible Astrophysical Solver. I. Hydrodynamics and Self-gravity, *Astrophys. J.* 715 (2010) 1221. arXiv:1005.0114, doi:10.1088/0004-637X/715/2/1221.
- [13] F. Löffler, J. Faber, E. Bentivegna, T. Bode, P. Diener, et al., The Einstein Toolkit: A Community Computational Infrastructure for Relativistic Astrophysics, *Class. Quantum Grav.* 29 (2012) 115001. arXiv:1111.3344, doi:10.1088/0264-9381/29/11/115001.
- [14] <http://www.black-holes.org/SpEC.html>.
- [15] P. R. Woodward, J. Jayaraj, P.-H. Lin, M. Knox, S. D. Hammond, J. Greensky, S. E. Anderson, Scaling the Multifluid PPM Code on Blue Waters and Intel MIC, in: 2013 Extreme Scaling Workshop (xsw 2013), IEEE, 2013, pp. 64–72. doi:10.1109/XSW.2013.13.
- [16] W. Reed, T. Hill, Triangular mesh methods for the neutron transport equation, Conference: National topical meeting on mathematical models and computational techniques for analysis of nuclear systems, Ann Arbor, Michigan, USA, 8 Apr 1973 LA-UR-73-479, CONF-730414-2 (1973). URL <http://www.osti.gov/scitech/servlets/purl/4491151>
- [17] J. Hesthaven, T. Warburton, *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*, Springer, Berlin, New York, 2008. doi:10.1007/978-0-387-72067-8.
- [18] B. Cockburn, Devising discontinuous Galerkin methods for non-linear hyperbolic conservation laws, *J. Comput. Appl. Math.* 128 (2001) 187. doi:10.1016/S0377-0427(00)00512-4.
- [19] B. Cockburn, C.-W. Shu, The Runge-Kutta Discontinuous Galerkin Method for Conservation Laws V. Multidimensional Systems, *J. Comput. Phys.* 141 (1998) 199. doi:10.1006/jcph.1998.5892.
- [20] B. Cockburn, An introduction to the Discontinuous Galerkin method for convection-dominated problems; in *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations: Lectures given at the 2nd Session of the Centro Internazionale Matematico Estivo (C.I.M.E.) held in Cetraro, Italy, June 23–28, 1997*, Lecture Notes in Physics, Springer, Berlin, New York, 1998, pp. 150–268. doi:10.1007/BFb0096353.
- [21] B. Cockburn, G. E. Karniadakis, C.-W. Shu, The Development of Discontinuous Galerkin Methods; in *Discontinuous Galerkin Methods: Theory, Computation and Applications*, in: B. Cockburn, G. E. Karniadakis, C.-W. Shu (Eds.), *Discontinuous Galerkin Methods: Theory, Computation and Applications*, Lecture Notes in Computational Science and Engineering, Springer, Berlin, New York, 2000, pp. 3–50. doi:10.1007/978-3-642-59721-3_1.
- [22] S. E. Field, J. S. Hesthaven, S. R. Lau, A. H. Mroue, Discontinuous Galerkin method for the spherically reduced BSSN system with second-order operators, *Phys. Rev. D* 82 (2010) 104051. arXiv:1008.1820, doi:10.1103/PhysRevD.82.104051.
- [23] J. D. Brown, P. Diener, S. E. Field, J. S. Hesthaven, F. Herrmann, A. H. Mroué, O. Sarbach, E. Schnetter, M. Tiglio, M. Wagman, Numerical simulations with a first-order BSSN formulation of Einstein’s field equations, *Phys. Rev. D* 85 (2012) 084004. arXiv:1202.1038, doi:10.1103/PhysRevD.85.084004.
- [24] S. E. Field, J. S. Hesthaven, S. R. Lau, Discontinuous Galerkin method for computing gravitational waveforms from extreme mass ratio binaries, *Class. Quantum Grav.* 26 (2009) 165010. arXiv:0902.1287, doi:10.1088/0264-9381/26/16/165010.

- [25] G. Zumbusch, Finite element, discontinuous Galerkin, and finite difference evolution schemes in space-time, *Class. Quantum Grav.* 26 (2009) 175011. [arXiv:0901.0851](#), [doi:10.1088/0264-9381/26/17/175011](#).
- [26] D. Radice, L. Rezzolla, Discontinuous Galerkin methods for general-relativistic hydrodynamics: formulation and application to spherically symmetric spacetimes, *Phys. Rev. D* 84 (2011) 024010. [arXiv:1103.2426](#), [doi:10.1103/PhysRevD.84.024010](#).
- [27] P. Mocz, M. Vogelsberger, D. Sijacki, R. Pakmor, L. Hernquist, A discontinuous Galerkin method for solving the fluid and magnetohydrodynamic equations in astrophysical simulations, *Mon. Not. Roy. Astr. Soc.* 437 (2014) 397. [arXiv:1305.5536](#), [doi:10.1093/mnras/stt1890](#).
- [28] O. Zanotti, M. Dumbser, A. Hidalgo, D. Balsara, An ADER-WENO finite volume AMR code for astrophysics, in: N. V. Pogorelov, E. Audit, G. P. Zank (Eds.), 8th International Conference of Numerical Modeling of Space Plasma Flows (ASTRONUM 2013), Vol. 488 of Astronomical Society of the Pacific Conference Series, 2014, pp. 285–290. [arXiv:1401.6448](#).
- [29] E. Endeve, C. D. Hauck, Y. Xing, A. Mezzacappa, Bound-preserving discontinuous Galerkin methods for conservative phase space advection in curvilinear coordinates, *J. Comput. Phys.* 287 (2015) 151. [arXiv:1410.7431](#), [doi:10.1016/j.jcp.2015.02.005](#).
- [30] K. Schaal, A. Bauer, P. Chandrashekar, R. Pakmor, C. Klingenberg, V. Springel, Astrophysical hydrodynamics with a high-order discontinuous Galerkin scheme and adaptive mesh refinement, *Mon. Not. Roy. Astr. Soc.* 453 (2015) 4278. [arXiv:1506.06140](#), [doi:10.1093/mnras/stv1859](#).
- [31] S. A. Teukolsky, Formulation of discontinuous Galerkin methods for relativistic astrophysics, *J. Comput. Phys.* 312 (2016) 333. [arXiv:1510.01190](#), [doi:10.1016/j.jcp.2016.02.031](#).
- [32] M. Bugner, T. Dietrich, S. Bernuzzi, A. Weyhausen, B. Bruegmann, Solving 3D relativistic hydrodynamical problems with WENO discontinuous Galerkin methods, Submitted to *Phys. Rev. D*, [arXiv:1508.07147](#).
- [33] O. Zanotti, F. Fambri, M. Dumbser, Solving the relativistic magnetohydrodynamics equations with ADER discontinuous Galerkin methods, a posteriori subcell limiting and adaptive mesh refinement, *Mon. Not. Roy. Astr. Soc.* 452 (2015) 3010. [arXiv:1504.07458](#), [doi:10.1093/mnras/stv1510](#).
- [34] J. M. Miller, E. Schnetter, An Operator-Based Local Discontinuous Galerkin Method Compatible With the BSSN Formulation of the Einstein Equations, Submitted to *Phys. Rev. D*, [arXiv:1604.00075](#).
- [35] M. Schaller, P. Gonnet, A. B. G. Chalk, P. W. Draper, SWIFT: Using task-based parallelism, fully asynchronous communication, and graph partition-based domain decomposition for strong scaling on more than 100,000 cores, in: *Proceedings of the Platform for Advanced Scientific Computing Conference*, 2016, 2, 2:1-2:10, Lausanne, Switzerland, 2016. [arXiv:1606.02738](#), [doi:10.1145/2929908.2929916](#).
- [36] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kal, K. Schulten, Scalable molecular dynamics with NAMD, *J. Comp. Chem.* 26 (2005) 1781. [doi:10.1002/jcc.20289](#).
- [37] <http://charm.cs.illinois.edu/OpenAtom/>.
- [38] P. Jetley, F. Gioachin, C. Mendes, L. V. Kale, T. Quinn, Massively parallel cosmological simulations with changa, in: *Parallel and Distributed Processing*, 2008. IPDPS 2008. IEEE International Symposium on Parallel & Distributed Processing, 2008, pp. 1–12. [doi:10.1109/IPDPS.2008.4536319](#).
- [39] <http://uintah.utah.edu>.

- [40] M. Berzins, J. Luitjens, Q. Meng, T. Harman, C. A. Wight, J. R. Peterson, Uintah: A Scalable Framework for Hazard Analysis, in: Proceedings of the 2010 TeraGrid Conference, TG '10, ACM, New York, NY, USA, 2010, pp. 3:1–3:8. doi:10.1145/1838574.1838577.
- [41] G. Martyna, E. Bohm, R. Venkataraman, A. Arya, L. V. Kalé, A. Bhatele, OpenAtom: Ab initio Molecular Dynamics for Petascale Platforms, Tech. Rep. LLNL-BOOK-608553, Lawrence Livermore National Laboratory (LLNL), Livermore, CA (2012).
URL <https://e-reports-ext.llnl.gov/pdf/708673.pdf>
- [42] <http://http://charm.cs.illinois.edu/research/charm>.
- [43] L. V. Kalé, S. Krishnan, CHARM++: A Portable Concurrent Object Oriented System Based on C++, in: Proceedings of the Eighth Annual Conference on Object-oriented Programming Systems, Languages, and Applications, OOPSLA '93, ACM, New York, NY, USA, 1993, pp. 91–108. doi:10.1145/165854.165874.
- [44] W. Shu, L. V. Kalé, Chare Kernel—a runtime support system for parallel computations, J. Par. Dist. Comp. 11 (1991) 198. doi:10.1016/0743-7315(91)90044-A.
- [45] L. V. Kalé, A. Bhatele, Parallel Science and Engineering Applications: The Charm++ Approach, CRC Press, Boca Raton, 2016.
- [46] T. W. Baumgarte, S. L. Shapiro, Numerical Relativity: Solving Einstein's Equations on the Computer, Cambridge University Press, New York, 2010. doi:10.1080/00107514.2011.586052.
- [47] L. Rezzolla, O. Zanotti, Relativistic Hydrodynamics, Oxford University Press, New York, 2013. doi:10.1093/acprof:oso/9780198528906.001.0001.
- [48] J. A. Font, Numerical hydrodynamics and magnetohydrodynamics in general relativity, Living Reviews in Relativity 11 (7). doi:10.1007/lrr-2008-7.
- [49] F. Galeazzi, W. Kastaun, L. Rezzolla, J. A. Font, Implementation of a simplified approach to radiative transfer in general relativity, Phys. Rev. D 88 (2013) 064009. arXiv:1306.4953, doi:10.1103/PhysRevD.88.064009.
- [50] A. Dedner, F. Kemm, D. Kröner, C.-D. Munz, T. Schnitzer, M. Wesenberg, Hyperbolic Divergence Cleaning for the MHD Equations, J. Comput. Phys. 175 (2) (2002) 645. doi:10.1006/jcph.2001.6961.
- [51] C. Palenzuela, L. Lehner, O. Reula, L. Rezzolla, Beyond ideal MHD: towards a more realistic modelling of relativistic astrophysical plasmas, Mon. Not. Roy. Astr. Soc. 394 (2009) 1727. arXiv:0810.1838, doi:10.1111/j.1365-2966.2009.14454.x.
- [52] A. J. Penner, General relativistic magnetohydrodynamic Bondi-Hoyle accretion, Mon. Not. Roy. Astr. Soc. 414 (2011) 1467. arXiv:1011.2976, doi:10.1111/j.1365-2966.2011.18480.x.
- [53] P. Mösta, B. C. Mundim, J. A. Faber, R. Haas, S. C. Noble, T. Bode, F. Löffler, C. D. Ott, C. Reisswig, E. Schnetter, GRHydro: a new open-source general-relativistic magnetohydrodynamics code for the Einstein toolkit, Class. Quantum Grav. 31 (2014) 015005. arXiv:1304.5544, doi:10.1088/0264-9381/31/1/015005.
- [54] W. I. Newman, N. D. Hamlin, Primitive variable determination in conservative relativistic magnetohydrodynamic simulations, SIAM Journal on Scientific Computing 36 (4) (2014) B661. doi:10.1137/140956749.
- [55] D. A. Kopriva, Implementing spectral methods for partial differential equations: Algorithms for scientists and engineers, Springer, Berlin, New York, 2009. doi:10.1007/978-90-481-2261-5.

- [56] A. G. Kulikovskii, N. V. Pogorelov, A. Y. Semenov, *Mathematical Aspects of Numerical Solution of Hyperbolic Systems*, CRC Press, Boca Raton, 2000.
- [57] M. Alcubierre, *Introduction to 3+1 Numerical Relativity*, Oxford University Press, New York, 2008.
- [58] C. F. Gammie, J. C. McKinney, G. Tóth, HARM: A Numerical Scheme for General Relativistic Magnetohydrodynamics, *Astrophys. J.* 589 (2003) 444. [arXiv:astro-ph/0301509](#), doi:10.1086/374594.
- [59] A. Harten, P. D. Lax, B. van Leer, On Upstream Differencing and Godunov-Type Schemes for Hyperbolic Conservation Laws, *SIAM Review* 25 (1983) 35. doi:10.1137/1025002.
- [60] E. F. Toro, *Riemann solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*, Springer, Berlin, New York, 2013. doi:10.1007/b79761.
- [61] S. F. Davis, Simplified Second-Order Godunov-Type Methods, *SIAM Journal on Scientific and Statistical Computing* 9 (1988) 445. doi:10.1137/0909030.
- [62] P. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, *J. Comput. Phys.* 43 (1981) 357. doi:[http://dx.doi.org/10.1016/0021-9991\(81\)90128-5](http://dx.doi.org/10.1016/0021-9991(81)90128-5).
- [63] P. L. Roe, J. Pike, Efficient Construction and Utilisation of Approximate Riemann Solutions, in: *Proc. Of the Sixth Int’l. Symposium on Computing Methods in Applied Sciences and Engineering, VI*, North-Holland Publishing Co., Amsterdam, 1985, pp. 499–518.
- [64] P. Glaister, An approximate linearised Riemann solver for the Euler equations for real gases, *J. Comput. Phys.* 74 (1988) 382. doi:10.1016/0021-9991(88)90084-8.
- [65] R. Donat, A. Marquina, Capturing Shock Reflections: An Improved Flux Formula, *J. Comput. Phys.* 125 (1996) 42. doi:10.1006/jcph.1996.0078.
- [66] J. M. Ibáñez, M. A. Aloy, J. A. Font, J. M. Martí, J. A. Miralles, J. A. Pons, *Riemann Solvers in General Relativistic Hydrodynamics*, Springer US, Berlin, New York, 2001, pp. 485–496. [arXiv:astro-ph/9911034](#), doi:10.1007/978-1-4615-0663-8_48.
- [67] S. A. Moe, J. A. Rossmann, D. C. Seal, A simple and effective high-order shock-capturing limiter for discontinuous galerkin methods.
- [68] J. Qiu, C.-W. Shu, A Comparison of Troubled-Cell Indicators for Runge–Kutta Discontinuous Galerkin Methods Using Weighted Essentially Nonoscillatory Limiters, *SIAM Journal on Scientific Computing* 27 (2005) 995. doi:10.1137/04061372X.
- [69] B. Cockburn, Discontinuous Galerkin methods for convection-dominated problems, in: *High-order Methods for Computational Physics*, Springer, Berlin, New York, 1999, pp. 69–224. doi:10.1007/978-3-662-03882-6_2.
- [70] B. Van Leer, Towards the ultimate conservative difference scheme. II. Monotonicity and conservation combined in a second-order scheme, *J. Comput. Phys.* 14 (1974) 361. doi:10.1016/0021-9991(74)90019-9.
- [71] B. Van Leer, Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov’s method, *J. Comput. Phys.* 32 (1979) 101. doi:10.1016/0021-9991(79)90145-1.
- [72] S. Gottlieb, C.-W. Shu, Total Variation Diminishing Runge-Kutta Schemes, *Math. Comput.* 67 (1998) 73. doi:10.1090/S0025-5718-98-00913-2.
- [73] S. Gottlieb, C.-W. Shu, E. Tadmor, Strong Stability-Preserving High-Order Time Discretization Methods, *SIAM Review* 43 (2001) 89. doi:10.1137/S003614450036757X.

- [74] H. Kaiser, M. Brodowicz, T. Sterling, ParalleX: an advanced parallel execution model for scaling-impaired applications, in: International Conference on Parallel Processing Workshops (2009 - Los Alamos, California - page 394 to 401), 2009.
- [75] T. Sterling, D. Kogler, M. Anderson, M. Brodowicz, SLOWER: A performance model for Exascale computing, *Supercomputing Frontiers and Innovations* 1 (2014) 42–57.
- [76] A. Gürsoy, L. V. Kalé, Performance and modularity benefits of message-driven execution, *J. Par. Dist. Comp.* 64 (2004) 461. doi:10.1016/j.jpdc.2004.03.006.
- [77] G. Zheng, E. Meneses, A. Bhatele, L. V. Kalé, Hierarchical load balancing for Charm++ applications on large supercomputers, in: 2010 39th International Conference on Parallel Processing Workshops, IEEE, 2010, p. 436. doi:10.1109/ICPPW.2010.65.
- [78] M. Bhandarkar, L. V. Kalé, E. de Sturler, J. Hoeflinger, Adaptive load balancing for MPI programs, in: International Conference on Computational Science, San Francisco, CA, USA, May 28—30, 2001, Proceedings, Part II, Springer, Berlin, New York, 2001, pp. 108–117. doi:10.1007/3-540-45718-6_13.
- [79] R. Alverson, D. Roweth, L. Kaplan, The Gemini System Interconnect, in: 2010 18th IEEE Symposium on High Performance Interconnects, 2010, pp. 83–87. doi:10.1109/HOTI.2010.23.
- [80] Y. Sun, G. Zheng, L. V. Kalé, T. R. Jones, R. Olson, A uGNI-based Asynchronous Message-driven Runtime System for Cray Supercomputers with Gemini Interconnect, in: Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, 2012, pp. 751–762. doi:10.1109/IPDPS.2012.127.
- [81] L. V. Kalé, G. Zheng, C. W. Lee, S. Kumar, Scaling Applications to Massively Parallel Machines Using Projections Performance Analysis Tool, *Future Gener. Comput. Syst.* 22 (2006) 347. doi:10.1016/j.future.2004.11.020.
- [82] M. Bugner, private communication (2016).
- [83] C. Gheller, A high resolution hydrodynamical code for the study of cosmological structures, Ph.D. thesis, SISSA, Trieste, Italy (1997).
URL <http://hdl.handle.net/1963/5607>
- [84] F. Lora-Clavijo, J. Cruz-Perez, F. S. Guzmán, J. Gonzalez, Exact solution of the 1D riemann problem in Newtonian and relativistic hydrodynamics, *Revista mexicana de física E* 59 (2013) 28.
- [85] L. Rezzolla, O. Zanotti, An improved exact Riemann solver for relativistic hydrodynamics, *J. Fluid Mech.* 449 (2001) 395. arXiv:gr-qc/0103005, doi:10.1017/S0022112001006450.
- [86] A. Lucas-Serrano, J. A. Font, J. M. Ibáñez, J. M. Martí, Assessment of a high-resolution central scheme for the solution of the relativistic hydrodynamics equations, *Astron. Astrophys.* 428 (2004) 703. arXiv:astro-ph/0407541, doi:10.1051/0004-6361:20035731.
- [87] W. Zhang, A. I. MacFadyen, RAM: A Relativistic Adaptive Mesh Refinement Hydrodynamics Code, *Astrophys. J. Suppl. Ser.* 164 (2006) 255. arXiv:astro-ph/0505481, doi:10.1086/500792.
- [88] C. D. Muhlberger, F. H. Nouri, M. D. Duez, F. Foucart, L. E. Kidder, et al., Magnetic effects on the low- $T/|W|$ instability in differentially rotating neutron stars, *Phys. Rev. D* 90 (2014) 104014. arXiv:1405.2144, doi:10.1103/PhysRevD.90.104014.
- [89] S. S. Komissarov, A Godunov-type scheme for relativistic magnetohydrodynamics, *Mon. Not. Roy. Astr. Soc.* 303 (1999) 343. doi:10.1046/j.1365-8711.1999.02244.x.

- [90] H. Yee, N. Sandham, M. Djomehri, Low-Dissipative High-Order Shock-Capturing Methods Using Characteristic-Based Filters, *J. Comput. Phys.* 150 (1999) 199. doi:10.1006/jcph.1998.6177.
- [91] Z. Xu, Y. Liu, C.-W. Shu, Hierarchical reconstruction for discontinuous Galerkin methods on unstructured grids with a WENO-type linear reconstruction and partial neighboring cells, *J. Comput. Phys.* 228 (2009) 2194. doi:10.1016/j.jcp.2008.11.025.
- [92] L. Del Zanna, O. Zanotti, N. Bucciantini, P. Londrillo, ECHO: a Eulerian conservative high-order scheme for general relativistic magnetohydrodynamics and magnetodynamics, *Astron. Astrophys.* 473 (2007) 11. arXiv:0704.3206, doi:10.1051/0004-6361:20077093.
- [93] S. A. Orszag, C.-M. Tang, Small-scale structure of two-dimensional magnetohydrodynamic turbulence, *J. Fluid Mech.* 90 (1979) 129. doi:10.1017/S002211207900210X.
- [94] D. Ryu, T. W. Jones, A. Frank, Numerical Magnetohydrodynamics in Astrophysics: Algorithm and Tests for Multidimensional Flow, *Astrophys. J.* 452 (1995) 785. arXiv:astro-ph/9505073, doi:10.1086/176347.
- [95] W. Dai, P. R. Woodward, On the Divergence-free Condition and Conservation Laws in Numerical Simulations for Supersonic Magnetohydrodynamical Flows, *Astrophys. J.* 494 (1998) 317. doi:10.1086/305176.
- [96] P. Londrillo, L. Del Zanna, High-Order Upwind Schemes for Multidimensional Magnetohydrodynamics, *Astrophys. J.* 530 (2000) 508. arXiv:astro-ph/9910086, doi:10.1086/308344.
- [97] K. Beckwith, J. M. Stone, A Second-order Godunov Method for Multi-dimensional Relativistic Magnetohydrodynamics, *Astrophys. J. Suppl. Ser.* 193 (2011) 6. arXiv:1101.3573, doi:10.1088/0067-0049/193/1/6.
- [98] F. C. Michel, Accretion of Matter by Condensed Objects, *Astrophysics and Space Science* 15 (1972) 153. doi:10.1007/BF00649949.
- [99] F. Pretorius, Numerical relativity using a generalized harmonic decomposition, *Class. Quantum Grav.* 22 (2005) 425. arXiv:gr-qc/0407110, doi:10.1088/0264-9381/22/2/014.
- [100] C. Gundlach, J. M. Martin-Garcia, G. Calabrese, I. Hinder, Constraint damping in the Z4 formulation and harmonic gauge, *Class. Quantum Grav.* 22 (2005) 3767. arXiv:gr-qc/0504114, doi:10.1088/0264-9381/22/17/025.
- [101] L. Lindblom, M. A. Scheel, L. E. Kidder, R. Owen, O. Rinne, A New Generalized Harmonic Evolution System, *Class. Quantum Grav.* 23 (2006) S447. arXiv:gr-qc/0512093v3, doi:10.1088/0264-9381/23/16/S09.
- [102] R. Haas, C. D. Ott, B. Szilagyi, J. D. Kaplan, J. Lippuner, M. A. Scheel, K. Barkett, C. D. Muhlberger, T. Dietrich, M. D. Duez, F. Foucart, H. P. Pfeiffer, L. E. Kidder, S. A. Teukolsky, Simulations of inspiraling and merging double neutron stars using the Spectral Einstein Code, *Phys. Rev. D* 93 (12) (2016) 124062. arXiv:1604.00782, doi:10.1103/PhysRevD.93.124062.
- [103] A. Dubey, A. Almgren, J. Bell, M. Berzins, S. Brandt, G. Bryan, P. Colella, D. Graves, M. Lijewski, F. Löffler, B. O'Shea, E. Schnetter, B. V. Straalen, K. Weide, A survey of high level frameworks in block-structured adaptive mesh refinement packages, *J. Parallel Distrib. Comput.* 74 (2014) 3217. doi:10.1016/j.jpdc.2014.07.001.
- [104] G. Gassner, D. A. Kopriva, A comparison of the dispersion and dissipation errors of Gauss and Gauss-Lobatto discontinuous Galerkin spectral element methods, *SIAM Journal on Scientific Computing* 33 (2011) 2560. doi:10.1137/100807211.
- [105] S. A. Teukolsky, Short note on the mass matrix for Gauss-Lobatto grid points, *Journal of Computational Physics* 283 (2015) 408. arXiv:1412.2276, doi:10.1016/j.jcp.2014.12.012.